ICE, CLOUD, and Land Elevation Satellite (ICESat) Project

GLAS_HDF Detailed Design

Revision -
November 1, 2012

SGT/Jeffrey Lee
Cryospheric Sciences Laboratory
Hydrospheric and Biospheric Processes
NASA Goddard Space Flight Center

Goddard Space Flight Center

National Aeronautics and
Space Administration

Table of Contents

List of Figures

List of Tables

1.0 INTRODUCTION

ICESat (Ice, Cloud, and land Elevation Satellite) was the benchmark Earth Observing System mission for measuring ice sheet mass balance, cloud and aerosol heights, as well as land topography and vegetation characteristics. From 2003 to 2009, the ICESat mission provided multi-year elevation data needed to determine ice sheet mass balance as well as cloud property information, especially for stratospheric clouds common over polar areas. It also provided topography and vegetation data around the globe, in addition to the polar-specific coverage over the Greenland and Antarctic ice sheets.

The GEOSCIENCE LASER ALTIMETER SYSTEM (GLAS) was the primary instrument aboard ICESat. GLAS was a laser altimeter that determined the distance from the satellite to the Earth's surface and to intervening clouds and aerosols by precisely measuring the time it takes for a short pulse of laser light to travel to the reflecting object and return to the satellite.

1.1 Identification of Document

This document is identified as the Detailed Design Document that describes the software that converts GLAS Level 1-2 integer-binary products (GLAS_BIN) into HDF5 format (GLAS_HDF).

1.2 Scope

This document describes the GLAS_HDF processing software. Original GLAS products (GLAS_BIN) were created in an integer-binary format. The GLAS_HDF software converts the GLAS_BIN products into HDF5 format in order to make the products more interoperable with future ICESat-2 products and to provide a testbed for designing and creating products in standards-compliant format.

This document focuses on the GLAS_HDF product generation executables (PGEs) that perform the actual transformation. It does not describe SDMS (Scheduling and Data Management System), the surrounding middleware that was re-used from I-SIPS (ICESat Science Investigator-led Processing System) to manage data availability, cataloging, job-control, and data transfer functions.

1.3 Purpose and Objectives

The purpose of this document is to describe the GLAS_HDF software. It states the requirements of the effort and traces those requirements to actual implementation. Since there are 15 nearly identical PGEs, this document details the software constructs as models in a fairly generic fashion, identifying specific PGEs only when a major deviation from the model is made.

The objectives of this document are to serve as a reference source which would assist the maintenance programmer in making changes which fix or enhance the documented

software; to provide a reference for other programmers attempting to reuse the whole or parts of the software; and serve as a guide for others creating similar software.

1.4 Acknowledgements

The following individuals/organizations contributed to this effort:

- ICESat GLAS and ICESat-2 ATLAS Science Software Development Teams

  SGT/Jeffrey Lee, SGT/John DiMarzio, SGT/Peggy Jester, SGT/Suneel Bhardwaj, SSAI/Kristine Barbieri, SGT/LeeAnne Roberts, Sigma/David W Hancock, SGT/Thomas Feroli, Sigma/Vijay Suchdeo, SGT/Andrew Griffin

- ESDIS

  423/Jeanne Behnke, 432/Jeff Walter, 423/Hampapuram Ramapriyan

- Earth Science Data Systems Working Groups

  Standards Process Group

  Technical Infusion Working Group

- NSIDC (ICESat Data Center)

  Siri Jodha Khalsa, Doug Fowler

- SMAP Product Development Team

  JPL/Barry Weiss

1.5 Document Status and Schedule

No further updates to this document are planned.

1.6 Document Change History

Revision	Date	Nature of Change
-	November 1, 2012	Original Version

1.7 Documentation Conventions

Within this document several documentation conventions are used to either strongly or generically identify items.

Term	Explanation
HDF5_LIBRARY	Refers to the software, a tool or a capability build into the HDF5 library provided by hdfgroup.org.
GLAS_HDF	Contextually refers to either the GLAS_HDF effort or the GLAS_HDF product set.

Term	Explanation
GLAS_BIN	Refers to the GLAS integer-binary data products.
glaxx	Refers to a specific GLAS integer-binary file type (where xx is 01-15)
glahxx	Refers to a specific GLAS HDF5 file type (where xx is 01-15)

2.0 RELATED DOCUMENTATION

2.1 Parent Documents

This document is subordinate to any top-level mission or instrument management plan documents, and as such, recognizes these documents as external parent documents in lineage. The recognized external EOSDIS and GLAS parent documents superior to this document are listed below.

NASA Earth Observing System Geoscience Laser Altimeter System GLAS Science Requirements Document, Version 2.01, October 1997, Center for Space Research, University of Texas at Austin.

GLAS Science Software Management Plan, NASA/TM-1999-208641/Version 3/ Volume 1, August 1998, NASA/GSFC Wallops Flight Facility.

2.2 Applicable Documents

The following documents are related to, or contain policies or references pertinent to the contents of this document.

GLAS_HDF Standard Data Products Specification, Version 9.1, August 2012, NASA Goddard Space Flight Center.

GLAS Standard Data Products Specification - Level 1, Version 9.0, August 2012, NASA Goddard Space Flight Center.

GLAS Standard Data Products Specification - Level 2, Version 9.0, August 2012, NASA Goddard Space Flight Center.

GLAS Standard Data Products Specification - Data Dictionary, Version1.0, August 2012, NASA Goddard Space Flight Center.

GLAS_HDF Standard Data Product, Revision -, November 1, 2012, NASA Goddard Space Flight Center

2.3 References

Table 2-1 contains a list of references found useful and/or authoritative.

Table 2-1 : References

Reference	Description
http://hdfgroup.org	HDF5 Documentation and examples.
http://glas.wff.nasa.gov/prod_format/v60_products	GLAS products data dictionary.
http://earthdata.nasa.gov/data/references/data-metadata-formats	NASA Data and Metadata format information.
http://science.nasa.gov/earth-science/earth-science-data/satellite-mission-data-system-requirements/	Satellite mission data requirements.

Reference	Description
http://earthdata.nasa.gov/our-community/esdswg/standards-process-spg/rfc	List of EOSDIS-approved standards.
http://earthdata.nasa.gov/our-community/esdswg/standards-process-spg/rfc/esds-rfc-007	EOSDIS HDF5 RFC
http://earthdata.nasa.gov/our-community/esdswg/standards-process-spg/rfc/esds-rfc-009-file-format-satellite-atmospheric-chemistry-data	EOSDIS Aura File Format Technical Note
http://earthdata.nasa.gov/our-community/esdswg/standards-process-spg/rfc/esds-rfc-021	EOSDIS CF Metadata Conventions RFC
http://earthdata.nasa.gov/our-community/esdswg/standards-process-spg/rfc/esds-rfc-022	EOSDIS NetCDF/HDF5 RFC
http://cf-pcmdi.llnl.gov/	CF Metadata
http://www.nodc.noaa.gov/data/formats/netcdf/	NOAA NODC NetCDF Templates

3.0 DRIVING REQUIREMENTS

The following driving requirements form the basis for GLAS_HDF software implementation.

Table 3-1 Driving Requirements

Identifier	Requirement
REQ_GLAS_HDF_001	The software shall transform GLAS integer-binary products into a standards compliant format.
REQ_GLAS_HDF_001.1	The software shall use HDF5 as the standard data product file format (ESDS-RFC-007).
REQ_GLAS_HDF_001.2	The software shall create the products with HDF5 CF-compliant parameter attributes to make the products self-documenting. This will allow data dictionaries to be created directly from the products themselves.
REQ_GLAS_HDF_001.3	The software shall create the products with NetCDF-compliance in mind. This may allow the products to be used with NetCDF/HDF tools.
REQ_GLAS_HDF_001.4	The software shall use compression where possible to decrease the size of the products.
REQ_GLAS_HDF_001.5	The software shall perform only transformation processes. No new science parameters shall be created.
REQ_GLAS_HDF_001.6	The software shall not put “spare” or un-implemented parameters on the products.
REQ_GLAS_HDF_002	The software shall make efforts to improve the usability of the products.
REQ_GLAS_HDF_002.1	The software shall create products in such a manner that individual data values may be independently read.
REQ_GLAS_HDF_002.2	The software shall logically group parameters, but at a level where desired data are not hidden.
REQ_GLAS_HDF_002.3	The software shall transform the parameters from scaled-integer units into scientific units.
REQ_GLAS_HDF_002.4	The software shall provide a mechanism whereby each instance of a parameter can be associated with a time stamp and a laser shot number.
REQ_GLAS_HDF_002.5	The software shall store variable-rate waveforms in volts and provide relative sample times that will enable easy decompression of the waveforms.
REQ_GLAS_HDF_002.6	The software shall incorporate multi-rate data within the same product.
REQ_GLAS_HDF_002.7	The software shall incorporate existing browse information into the products where available.
REQ_GLAS_HDF_002.8	The software shall unpack bit flags where existing unpack routine already exist.
REQ_GLAS_HDF_003	The software shall incorporate metadata into the products.
REQ_GLAS_HDF_003.1	The products will incorporate both human-readable and computer-parseable

Identifier	Requirement
	metadata.
REQ_GLAS_HDF_003.2	The software shall support the same method of metadata exchange with NSIDC as the GLAS_BIN products. (External .MET files in ECHO format.)
REQ_GLAS_HDF_003.3	The software will store lineage metadata on the products such that prior processing information is not lost.
REQ_GLAS_HDF_003.4	The software shall support product-level digital object identifiers (DOIs) as defined by the ESDIS pilot DOI effort.
REQ_GLAS_HDF_003.5	The software shall support UUIDs as granule-level unique identifiers to extend the ESDIS pilot DOI effort.
REQ_GLAS_HDF_004	The software shall re-use existing software to the maximum extent possible.
REQ_GLAS_HDF_004.1	The software shall re-use existing GLAS Science Algorithm Software (GSAS).
REQ_GLAS_HDF_004.2	The software shall re-use existing MABEL Science Algorithm Software.
REQ_GLAS_HDF_004.3	The software shall be written to interface with re-used I-SIPS SDMS middleware for data management and job control.

The required GLAS_HDF data products types are listed in Table 3-2.

Table 3-2 GLAS_HDF Product Types

Product ID	Product Name	Level
GLAH01	Altimetry Data File	1A
GLAH02	Atmosphere Data File	1A
GLAH03	Engineering Data File	1A
GLAH04	Combined LPA, LRS, GYRO, IST, BST, SPCA Data File	1A
GLAH05	Waveform-based Elevation Corrections File	1B
GLAH06	Elevation File	1B
GLAH07	Backscatter File	1B
GLAH08	Boundary Layer and Elevated Aerosol Layer Heights File	2
GLAH09	Cloud Height for Multiple Layers File	2
GLAH10	Aerosol Vertical Structure File	2
GLAH11	Thin Cloud/Aerosol Optical Depth File	2
GLAH12	Ice Sheet Products File	2
GLAH13	Sea Ice Products File	2
GLAH14	Land Products File	2
GLAH15	Ocean Products File	2

4.0 ENVIRONMENT

The GLAS_HDF software is developed within the re-used I-SIPS environment. The GLAS_HDF effort is the third-reuse of I-SIPS and the accompanying SDMS software. The second reuse is an ongoing MABEL processing system. MABEL is the airborne demonstrator instrument for the ICESat-2 photon-counting LIDAR.

4.1 Hardware

The I-SIPS environment consists of Linux-based x86 hardware. Sufficient computing and storage resources exist to handle both development and execution of the software system.

4.2 Tools

The MABEL effort is a prototype for the ICESat-2 Atlas Science Algorithm Software (ASAS) and several new tools were adopted with an outlook towards ICESat-2 software development. The GLAS_HDF effort has incorporated those new tools as well. Tools used in GLAS_HDF development are listed in Table 4-1.

Table 4-1 Tools

Category	Tool
Development (compiler)	gfortran 4.2.x
Development (tool)	IDL 8.0
Library	HDF5 1.8.9 (HDF5_LIBRARY)
SCM	AccuRev
Issue Tracking	Atlassian Jira
Wiki	Atlassian Confluence

In addition, many of the software management processes planned for ICESat-2 were adopted for this development effort.

5.0 SOFTWARE ARCHITECTURE

The GLAS_HDF software is comprised of several architectural components. Some of these are re-used legacy code; some are newly developed; and others are machine-generated. Two types of PGEs are created for each product type: a data converter and a data dictionary creator. Since each respective data conversion and data dictionary PGE is nearly identical in structure, a generic PGE will be used to describe the architecture. Three additional PGEs were created to 1) create the GLAS_HDF product APIs, 2) embed the browse products and 3) generate detached metadata files.

5.1 Architectural Components

GLAS_HDF architectural components are listed in Table 5-1. Where "xx" is used, an individual PGE exists for each product type.

Table 5-1 Architectural Components

Name	Description	Source
common_lib	Common Library	reused & improved from MABEL/GSAS
gsas_lib	GSAS Product Library	reused from GSAS
glashdf_lib	GLAS_HDF-specific Library	developed
gla_codegen	GLAS_HDF API code generator	developed
glaxx_api	GLAS_HDF product APIs	generated/improved
glaxx_h5_convert	GLAS_HDF product conversion PGE	developed/generated
glaxx_dd	GLAS_HDF data dictionary generation PGE	generated/developed
glah_meta	GLAS_HDF metadata generation PGE	developed
glah_brw	GLAS_HDF browse PGE	developed (IDL)

Figure 5-1 illustrates the relationships between various components. Each component will be fully described in its respective detailed design section but briefly described here in its architectural relationship to other software components.

5.1.1 common_libs

common_lib is a library of generic Fortran routines that provide the base layer for the GLAS_HDF software. This library was originally re-used from GSAS and is being maintained and improved by both GLAS_HDF and MABEL development efforts. The library provides standard error-handling routines, control parsing routines, mathematical functions, text-handling routines and time conversion routines. The HDF5 and time routines are major additions from the legacy GSAS code. The HDF5 routines provide a standardized interface to the HDF5_LIBRARY. The time routines are a port of the HDF-EOS MDT Toolkit library that removes some of the implementation-specific EOSDIS conventions.

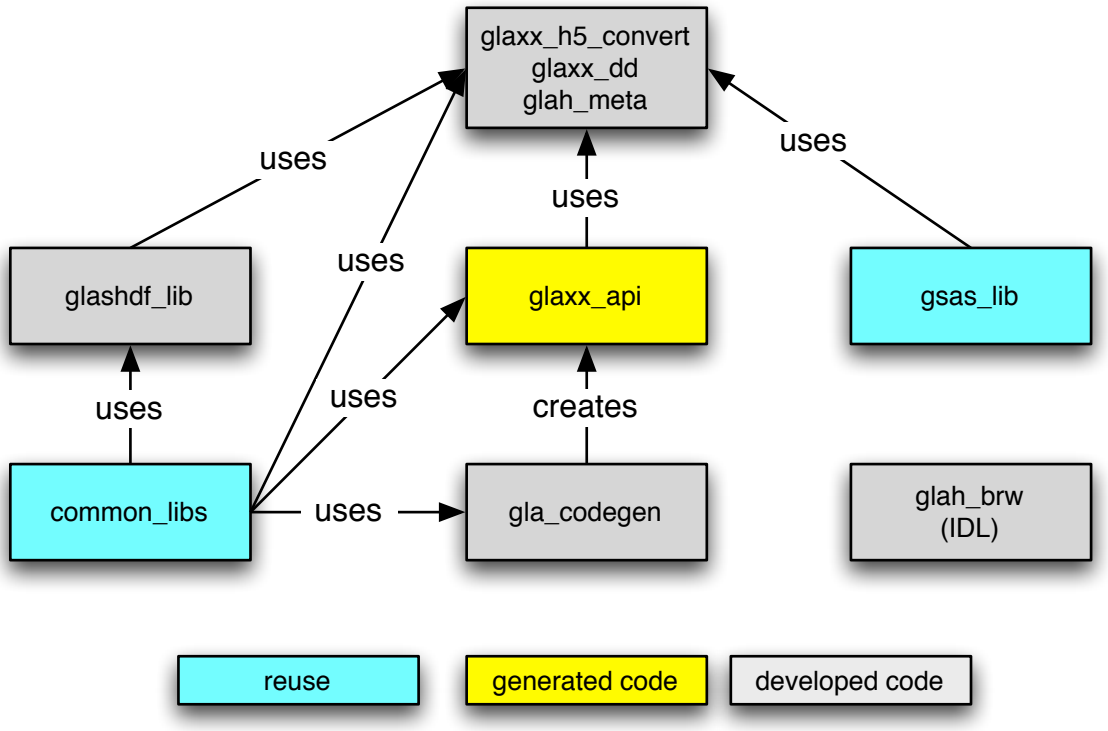

Figure 5-1 Architectural Relationship

5.1.2 gsas_lib

gsas_lib is a nearly-unchanged set of GSAS product routines that provide support functionality for the re-used routines that read, write and scale GSAS products. Additional support is provided for reading GLAS_BIN product headers and writing detached metadata files. This code was changed only to rectify minor compiler differences.

5.1.3 glashdf_lib

glashdf_lib is a collection of GLAS_HDF-specific routines that are shared by (and exclusive to) the GLAS_HDF PGEs. These routines are exclusively used to handle GLAS_HDF metadata.

5.1.4 gla_codegen

HDF5 moves the software burden from the user of a data product to the creator of the data product. The cost of implementing HDF5 in a standards-compliant manner introduces even more burden onto the data creator. To alleviate this burden, gla_codegen reads a GLAS data product description and creates PGEs and product APIs to automate most of the coding for the conversion and data-dictionary generation routines.

5.1.5 glahxx_api

An instance of glaxx_api exists for each product type. The initial code is generated by gla_codegen and then lightly modified by the developer to fully instantiate the API (application programming interface). Each API provides routines to create, read, write and document its respective product.

5.1.6 glaxx_h5_convert

An instance of glaxx_h5_convert exists for each product type. The initial code framework is generated by gla_codegen and then heavily modified by the developer to fully instantiate the PGE. The primary change required to the initial code is to incorporate multi-rate data. Since GLAS products contain extensive multi-rate data, this is a non-trivial change. Code to read and convert the GLAS_BIN products is reused from GSAS. Once developed, the PGE performs the complete conversion of an input GLAS_BIN product to GLAS_HDF.

5.1.7 glaxx_dd

An instance of glaxx_dd exists for each product type. The initial code is generated by glas_codegen and then lightly modified (if required) by the developer to fully instantiate the PGE. The PGE creates a HTML-based data dictionary from metadata and parameter attributes stored within a respective GLAS_HDF product type.

5.1.8 glah_meta

glas_meta is a developed PGE which reads an input EOSDIS ESDT file, parses metadata from any of the GLAS_HDF product types and creates an ECHO-style detached metadata file for ingest into a EOSDIS-based datacenter.

5.1.9 glah_brw

glah_brw is implemented as generic IDL code that reads a set of indexed color HDF4 images from a HDF4 file, transforms them to TrueColor and writes the transformed images to a “BROWSE” group in a HDF5 file.

6.0 FUNCTIONAL OVERVIEW

This section describes the functional operation of the GLAS_HDF software. Each PGE has been briefly described in the previous section. This section will detail the inputs and outputs of each PGE and illustrate the flow of data within the system. Since it is not part of the operational system, the gla_codegen process will be described in its own detailed design section. A high-level overview of the complete system is show in Figure 6-1.

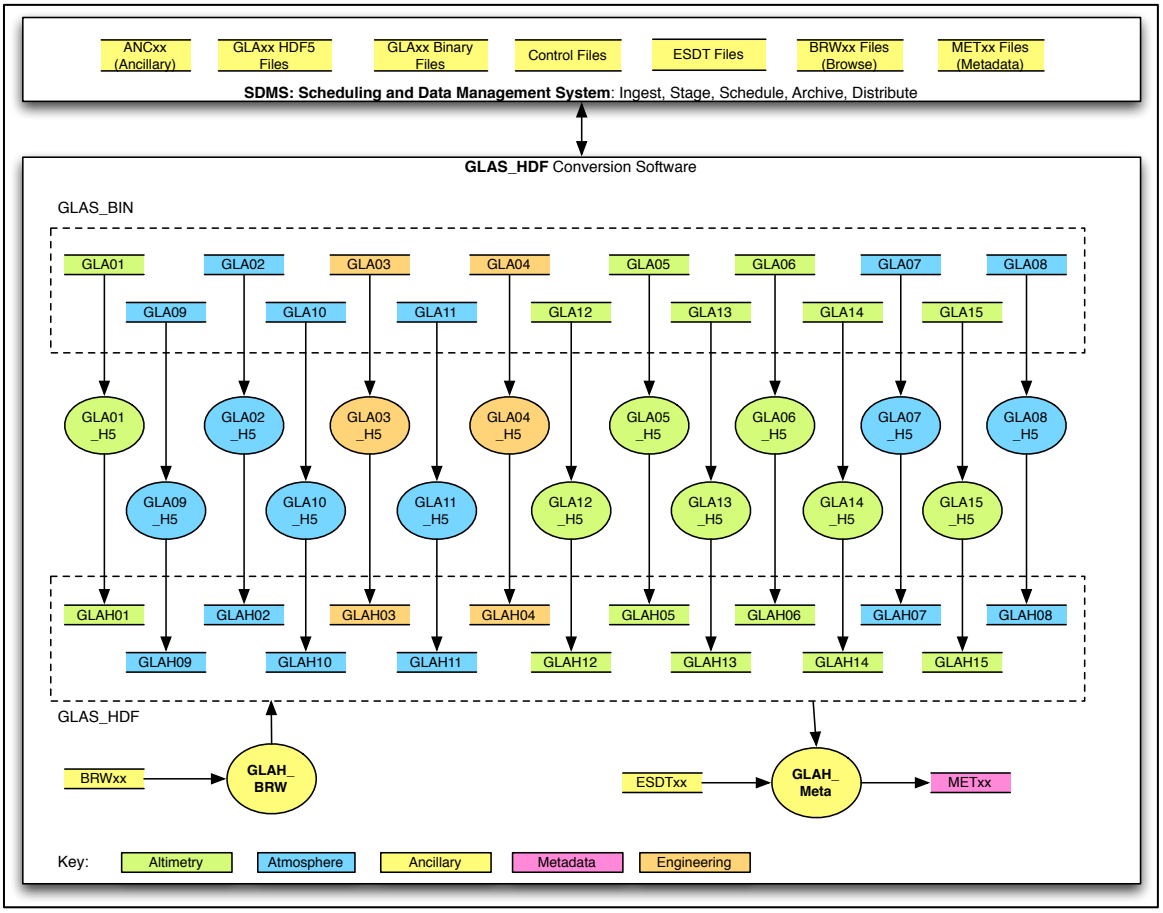

Figure 6-1 GLAS_HDF Overview

There are four primary processes within the GLAS_HDF software system.

- HDF5 Conversion
- Data Dictionary Generation
- Browse Attachment
- Detached Metadata Creation

Each process is instantiated as a single PGE. Figure 6-2 illustrates the flow of these processes as a comprehensive system.

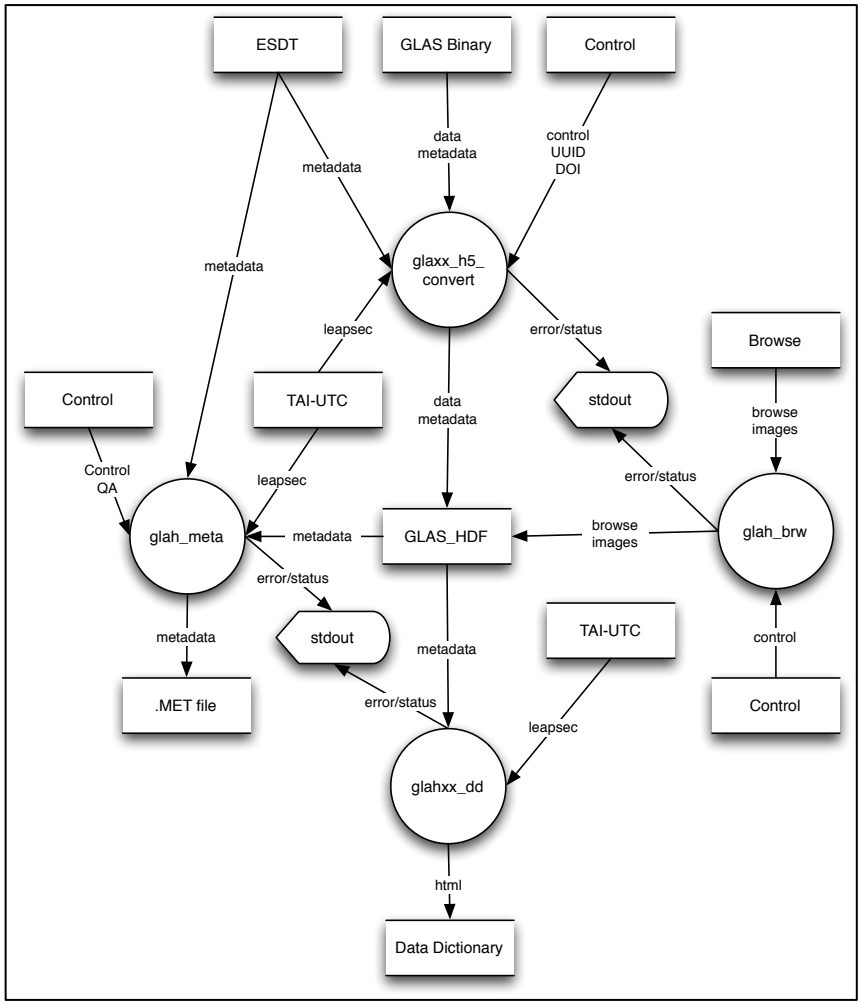

Figure 6-2 GLAS_HDF Dataflow

6.1 HDF5 Conversion Process

HDF5 conversion is the primary GLAS_HDF process. It uses the appropriate glaxx_h5_convert PGE to transform the GLAS_BIN granules into GLAS_HDF granules. The process is run for each GLAS_BIN granule to be converted.

6.1.1 Inputs/Outputs

Table 6-1 HDF5 Conversion Inputs

Input	Description
control	control file generated by I-SIPS SDMS that contains the names of input files, names of output files and any control overrides specified for processing.
TAI_UTC	TAI-UTC ancillary input file used for time conversion.
GLAS_BIN	GLAS integer-binary granule.

ESDT	EOSDIS-created ESDT file containing inventory and collection-level metadata.

Table 6-2 HDF5 Conversion Outputs

Outputs	Description
status/error messages	Status and/or error messages generated by the software (stdout).
GLAS_HDF	Granule converted to HDF5 format.
Result Code	Result code indicating success or failure.

6.1.2 Control

Execution instructions for the HDF5 conversion process are contained within a control file. The control file contains all the control information needed to execute the process. Control file conventions are described in the detailed design section of common_libs/cntl_lib. An example control file is listed below and an explanation of each line follows. Each control file entry is a single line. Ignore the line wrapping in both the example and explanation.

```
=GLA05_h5_convert
identifier_file_UUID=D65E7C2A-7BC1-444F-AE6F-991DAD0B45FF
IN_ESDT=./DsESDTGlGLAH05.033.desc 2008-03-03t02:51:46 2008-03-04t01:24:53 33 1
IN_ANC_TAIUTC=../data/tai-utc.dat  888547920.000000 888629107.000000 1 1
IN_GLA05=GLA05_633_2123_002_0141_1_01_0001.DAT 2008-03-03t02:51:46 2008-03-
03t03:13:12 33 1
OUT_GLAH05=GLAH05_633_2123_002_0141_1_01_0001.H5 2008-03-03t02:51:46 2008-03-
03t03:13:12 33 1
```

Table 6-3 glaxx_h5_convert Control

Control Line	Explanation
=GLAxx_h5_convert	Control section identifier designated by the name of the PGE.
identifier_file_UUID=UUID	UUID generated for each file converted. Uniquely identifies a file.
IN_ESDT=filename start stop [release] [version]	First argument designates the path and name of an input ESDT file that corresponds to the input GLAS_BIN file. The second and third arguments are the start of the data time period for which the ESDT file is valid. These times may be expressed as CCSDS-A UTC or gps_seconds. In practice, this time should be the GLAS_BIN granule start/stop time. Release [optional] is the release number of the ESDT file (normally the same as the GLAS_BIN release). Version [optional] is the version number of the ESDT file (normally the same as the GLAS_BIN version).
IN_ANC_TAIUTC=filename start stop [release] [version]	First argument designates the path and name of an input TAI-UTC file. The second and third arguments are the start of the data time period for which the TAI-UTC file is valid. These times must be expressed as gps_seconds since leap second information (contained within the TAI-UTC file) is required for conversion between UTC and GPS time. In practice, this time should be the same GLAS_BIN granule start/stop time.

Control Line	Explanation
	Release [optional] is the release number of the TAI-UTC. Version [optional] is the version number of the TAI-UTC.
IN_GLAxx=filename start stop [release] [version]	First argument designates the path and name of an input GLAS_BIN file to be converted. The second and third arguments are the start of the data time period for which the GLAS_BIN file is valid. These times may be expressed as CCSDS-A UTC or gps_seconds. In practice, this time should be the GLAS_BIN granule start/stop time. Release [optional] is the release number of the GLAS_BIN. Version [optional] is the version number of the GLAS_BIN.
OUT_GLAHxx=filename start stop [release] [version]	First argument designates the path and name of the output GLAS_HDF file. The second and third arguments are the start of the data time period for which the GLAS_HDF file is valid. These times may be expressed as CCSDS-A UTC or gps_seconds. In practice, this time should be the GLAS_BIN granule start/stop time. Release [optional] is the release number of the GLAS_HDF. Version [optional] is the version number of the GLAS_HDF.

All instances of glaxx_h5_convert share the same control file structure/information with the exception of the PGE for GLAH04. Since GLA04 is a multi-file granule that is converted into a single GLAH04 granule, each GLA04 sub-granule requires an additional IN_GLAxx line in the control. To specify each sub-granule, the keyword "IN_GLAxx" is changed to "IN_GLAxxyy" where yy is the sub-granule ID (01-06).

6.1.3 Process Initiation

The process is invoked by running the appropriate glaxx_h5_convert PGE with a valid control file as the sole command-line argument.

Example :

```
# gla06_h5_convert cf_20100305_19393.ctl
```

6.1.4 Error Detection Recovery

Error/status messages are displayed on stdout. Any errors should be reported to the software development team for analysis. Upon software termination, a result code is returned to the parent shell. 0 indicates successful execution; non-zero indicates a failure.

6.2 Data Dictionary Generation Process

The data dictionary generation process uses the appropriate glaxx_dd PGE to create a HTML-based data dictionary describing the respective product type. The process only needs to be run once for each representative product type.

6.2.1 Control

The only control information required/available for glaxx_dd is the path and name of a GLAS_HDF file passed as a command-line argument.

6.2.2 Process Initiation

The process is invoked by running the glahxx_dd PGE with a valid GLAS_HDF filename as the sole command-line argument.

Example :

```
# glah06_dd GLAH06_633_2123_002_0141_4_01_0001.H5
```

6.2.3 Inputs/Outputs

Table 6-4 Data Dictionary Generation Inputs

Input	Description
GLAS_HDF	GLAS_HDF product granule.

Table 6-5 Data Dictionary Generation Outputs

Outputs	Description
status/error messages	Status and/or error messages generated by the software (stdout).
GLAHxx_data_dict.html	HTML-based GLAS_HDF product data dictionary.
Result Code	Result code indicating success or failure

6.2.4 Error Detection Recovery

Error/status messages are displayed on stdout. Any errors should be reported to the software development team for analysis. Upon software termination, a result code is returned to the parent shell. 0 indicates successful execution; non-zero indicates a failure.

6.3 Browse Attachment Process

The browse attachment process uses IDL and the hdf2hdf5.pro program to copy browse images from an original GLAS_BRW browse product to a GLAS_HDF product. This process needs to be run for every granule for which the browse images are available. Since the browse images are attached to an existing GLAS_HDF file, the GLAS_HDF file is considered both an input and output.

6.3.1 Inputs/Outputs

Table 6-6 Browse Attachment Inputs

Input	Description
GLAS_HDF	GLAS_HDF product granule.
GLAS_BRW	GLAS_BIN browse product.

Table 6-7 Browse Attachment Outputs

Outputs	Description
status/error messages	Status and/or error messages generated by the software (stdout)
GLAS_HDF	GLAS_HDF product granule

6.3.2 Control

The only control information required/available for hdf2hdf5.pro are the names of corresponding GLAS_BRW and GLAS_HDF files (respectively).

6.3.3 Process Initiation

The process is invoked launching the IDL environment and then running the hdf2hdf5.pro program with valid GLAS_BRW and GLAS_HDF filenames as respective comma-separated arguments.

Example:

```
# idl
IDL> .rnew hdf2hdf5
IDL> hdf2hdf5, GLAH06_633_2123_002_0141_4_01_0001.H5,
GLA06_633_2123_002_0141_4_01_BRWS_0001.HDF
```

6.3.4 Error Detection Recovery

Error/status messages are displayed on stdout. Any errors should be reported to the software development team for analysis. Since this process is run within the IDL environment, no result code is set.

6.4 Detached Metadata Creation Process

The detached metadata creation process uses the glah_meta PGE to create a detached metadata files from an input GLAS_HDF granule. The detached metadata file is required for an EOSDIS data center to ingest the GLAS_HDF granule. This process needs to be run for each GLAS_HDF granule created.

6.4.1 Inputs/Outputs

Table 6-8 Detached Metadata Creation Inputs

Input	Description
GLAS_HDF	GLAS_HDF product granule
TAI_UTC	TAI-UTC ancillary input file used for time conversion.

ESDT	EOSDIS-created ESDT file containing inventory and collection-level metadata.

Table 6-9 Detached Metadata Creation Outputs

Outputs	Description
status/error messages	Status and/or error messages generated by the software (stdout)
MET	Detached metadata file.
Result Code	Result code indicating success or failure

6.4.2 Control

Execution instructions for the detached metadata creation process are contained within a control file. The control file contains all the control information needed to execute the process. Control file conventions are described in the detailed design section of common_libs/cntl_lib. An example control file is listed below and an explanation of each line follows. Each control file entry is a single line. Ignore the line wrapping in both the example and explanation.

```
=glah_meta
IN_ESDT=./DsESDTGlGLAH05.033.desc 2008-03-03t02:51:46 2008-03-04t01:24:53 33 1
IN_ANC_TAIUTC=../data/tai-utc.dat  888547920.000000 888629107.000000 1 1
IN_GLAH=GLAH05_633_2123_002_0141_1_01_0001.H5 2008-03-03t02:51:46 2008-03-
03t03:13:12 33 1
OUT_MET=GLAH05_633_2123_002_0141_1_01_0001.MET 2008-03-03t02:51:46 2008-03-
03t03:13:12 33 1
```

Table 6-10 glaxx_h5_convert Control

Control Line	Explanation
=glah_meta	Control section identifier designated by the name of the PGE.
IN_ESDT=filename start stop [release] [version]	First argument designates the path and name of an input ESDT file that corresponds to the input GLAS_HDF file. The second and third arguments are the start of the data time period for which the ESDT file is valid. These times may be expressed as CCSDS-A UTC or gps_seconds. In practice, this time should be the GLAS_HDF granule start/stop time. Release [optional] is the release number of the ESDT file (normally the same as the GLAS_HDF release). Version [optional] is the version number of the ESDT file (normally the same as the GLAS_HDF version).
IN_ANC_TAIUTC=filename start stop [release] [version]	First argument designates the path and name of an input TAI-UTC file. The second and third arguments are the start of the data time period for which the TAI-UTC file is valid. These times must be expressed as gps_seconds since leap second information (contained within the TAI-UTC file) is required for conversion between UTC and GPS time. In practice, this time should be the same GLAS_HDF granule start/stop time. Release [optional] is the release number of the TAI-UTC. Version [optional] is the version number of the TAI-UTC.

Control Line	Explanation
IN_GLAH=filename start stop [release] [version]	First argument designates the path and name of an input GLAS_HDF file to be converted. Note that the keyword does not change for different file types. glah_meta parses the filename to determine the file type. The second and third arguments are the start of the data time period for which the GLAS_HDF file is valid. These times may be expressed as CCSDS-A UTC or gps_seconds. In practice, this time should be the GLAS_HDF granule start/stop time. Release [optional] is the release number of the GLAS_HDF. Version [optional] is the version number of the GLAS_HDF.
OUT_MET=filename start stop [release] [version]	First argument designates the path and name of the output MET file. The second and third arguments are the start of the data time period for which the MET file is valid. These times may be expressed as CCSDS-A UTC or gps_seconds. In practice, this time should be the GLAS_HDF granule start/stop time. Release [optional] is the release number of the GLAS_HDF. Version [optional] is the version number of the GLAS_HDF.

All instances of glaxx_h5_convert share the same control file structure/information

Execution instructions for the detached metadata creation process are contained within a control file. The control file contains all the control information needed for the process. Example content includes input/output file specification and PGE-specific processing instructions. The control file format/content is defined in the detailed design section of glah_meta PGE.

6.4.3 Process Initiation

The process is invoked by running the glah_meta PGE with a valid control file as the sole command-line argument.

Example :

```
# glah_meta cf_20100305_19394.ctl
```

6.4.4 Error Detection Recovery

Error/status messages are displayed on stdout. Any errors should be reported to the software development team for analysis. Upon software termination, a result code is returned to the parent shell. 0 indicates successful execution; non-zero indicates a failure.

7.0 COMMON_LIBS COMPONENT

The base level of GLAS_HDF software is a collection of core library routines inherited from GSAS and reused by MABEL. These libraries are coded in a generic manner such that GLAS_HDF, MABEL, and other development efforts can make use of the library routines. This design maximizes code reuse and all inherent advantages. Table 7-1 lists each component of common_libs. Each library component will be detailed in following sub-sections. Each sub-section will list global variables and subroutines provided by the library component. When appropriate, important constructs instantiated by the library component will also be discusses.

Table 7-1 common_lib Libraries

Library	Function
const_lib	Contains global constants.
err_lib	Contains error-related constants. Provides subroutines for standardized error handling and a routine to override error parameters via control.
mutil_lib	Contains subroutines that provide utility functions such as text processing, HTML generation, keyword/value implementation, and file structure definition.
time_lib	Contains time-related constants. Provides subroutines that convert between various time standards (such as GPS, UTC, and Julian).
math_lib	Contains subroutines that provide standard mathematical functions (such as statistics generation and interpolation).
cntl_lib	Contains subroutines for parsing control information.
anc_lib	Contains subroutines for accessing ancillary data files (such as DEMs and ESDTs).
hdf_lib	Contains subroutines that provide an interface to the HDF5_LIBRARY for handling GLAS_HDF-like HDF5 files in a standardized manner.

7.1 const_lib

const_lib contains a single Fortran module and provides global constants and a routine to initialize selected global constants. const_lib is a direct-reuse of MABEL/GSAS code with GLAS-specific parameters removed.

7.1.1 Globals

Selected global constants are listed in Table 7-2 (not all constants provided are directly relevant to GLAS_HDF).

Table 7-2 const_lib Globals

Module	Variable	Description
const_glob_mod	MAXKEY	Keyword maximum length
const_glob_mod	MAXSTR	Maximum string length

Module	Variable	Description
const_glob_mod	MAXLINE	Maximum line length
const_glob_mod	VERS_LEN	Maximum length of version string
const_glob_mod	COMMON_LIB_NAME	name of common_libs
const_glob_mod	COMMON_LIB_VERS	version of common_libs
const_glob_mod	COMMON_LIB_DATE	date of common_libs
const_glob_mod	COMMON_LIB_INFO	description of common_libs
const_glob_mod	I1B_ROLL	1-byte rollover value
const_glob_mod	I2B_ROLL	2-byte rollover
const_glob_mod	I2B_ROLL_MAX	2-byte maximum value
const_glob_mod	I4B_ROLL	4-byte rollover
const_glob_mod	INVALID_R8B	Invalid value for double precision datatype
const_glob_mod	INVALID_R4B	Invalid value for real datatype
const_glob_mod	INVALID_I4B	Invalid value for 4-byte integer
const_glob_mod	INVALID_I2B	Invalid value for 2-byte integer
const_glob_mod	INVALID_I1B	Invalid value for 1-byte integer
const_glob_mod	gd_PI	Value of PI
const_glob_mod	gd_C	Value for speed of light (m/s)
const_glob_mod	g_time_sec	Contains the current time. This may be filled with either system time or data time, depending on implementation.

7.1.2 Subroutines

A single subroutine is provided by const_lib. This subroutine must be called at the start of any program that uses const_lib and is listed in Table 7-3.

Table 7-3 const_lib Subroutines

Module	Subroutine	Description
const_glob_mod	const_glob_init	Initializes PGE information and invalid values.

7.2 err_lib

err_lib contains a single Fortran module, which provides global error codes and subroutines that perform standardized error and status handling. These routines write error and status messages in a standard format, control the type of time printed in error/status messages, and terminate processing if an error is deemed fatal. err_lib is a direct reuse of MABEL code, which is a simplified version of GSAS error handling.

7.2.1 Globals

Critical global variables are listed in table. Individual error codes are too numerous to list here. Variables listed with "CONTROL OVERIDE" can be set via control.

Table 7-4 err_lib Globals

Module	Variable	Description
error_mod	GE_NOERROR	Indicates no error detected.
error_mod	GE_NOTICE	Indicates a notice was detected.
error_mod	GE_WARNING	Indicates a warning was detected.
error_mod	GE_FATAL	Indicates a fatal error was detected.
error_mod	ERRORUNIT	Unit where errors are written. CONTROL OVERRIDE
error_mod	STATUSUNIT	Unit where status messages are written. CONTROL OVERRIDE
error_mod	g_use_datatime	Flag to indicate if system or data time will be printed.
error_mod	STATUSLEVEL	Bitflag indicating the level of status messages to write. CONTROL OVERRIDE

7.2.2 Subroutines

err_lib subroutines are listed in Table 7-5.

Table 7-5 err_lib Subroutines

Module	Subroutine	Description
error_mod	check_error	Checks error code. Writes error message if detected. Exits program if required.
error_mod	status	Writes status message.
error_mod	print_start_banner	Writes PGE information at start of execution.
error_mod	print_end_banner	Writes processing information at end of execution.

7.3 mutil_lib

mutil contains several Fortran modules that provide subroutines that perform a variety of utility functions. These functions include text processing, keyword/value implementation, and file structure handling. mutil_lib is a direct reuse of MABEL code that is improved version of GSAS code.

7.3.1 Globals

mutil_lib global variables are listed in Table 7-6. Individual error codes are too numerous to list here. Variables listed with "CONTROL OVERIDE" can be set via control.

Table 7-6 mutil_lib Globals

Module	Variable	Description
fstruct_mod	fstruct_type	File structure (fstruct) type definition. An fstruct structure contains a variety of information related to a file.
keyval_mod	keyval_type	Keyword/Value (keyval) type definition. A keyval structure contains a keyword and value pair. This forms the basis of control file and metadata constructs.

7.3.2 Subroutines

mutil_lib subroutines are listed in Table 7-7.

Table 7-7 mutil_lib Subroutines

Module	Subroutine	Description
fstruct_mod	init_fstruct	Initializes an fstruct variable.
fstruct_mod	print_fstruct	Prints components of an fstruct variable.
fstruct_mod	get_filepath	Splits a file specification into path and filename.
fstruct_mod	cat_filepath	Concatenates a path and filename.
html_mod	write_css	Writes the embedded CSS structure used within an HTML data dictionary.
keyval_mod	init_keyval	Initializes a keyval variable
keyval_mod	parse_keyval	Parses a keyval from a text string.
keyval_mod	print_keyval	Prints components of a keyval.
keyval_mod	find_key	Finds the specified keyword within a keyval list.
keyval_mod	find_kval	Finds the specified value within a keyval list.
keyval_mod	count_keys	Counts the number of specified keys within a keyval list.
keyval_mod	count_kvals	Counts the number of specified values within a keyval list.
textutil_mod	readline	Read a line of text from a file, skipping empty lines and comments.
textutil_mod	compare_str	Compares two strings (case and whitespace insensitive).
textutil_mod	single_line	Returns 80 -
textutil_mod	double_line	Returns 80 =
textutil_mod	is_digit	Verifies character is a digit.
textutil_mod	strsplit	Splits a string via given delimiter.

Module	Subroutine	Description
textutil_mod	count_fields	Counts number of strings separated by a given delimiter.
textutil_mod	empty_string	Returns TRUE if a string contains only whitespace.
textutil_mod	strip_str	Returns portion of a string up to the first control character. (Useful for handling HDF5 C-style strings).
textutil_mod	text2upper	Converts text to upper case.
textutil_mod	text2lower	Converts text to lower case.
textutil_mod	strreplace	Replace all occurrences of a substring within a string.
textutil_mod	find_str	Finds a string with list of strings.

7.4 time_lib

time_lib contains several Fortran modules that provide subroutines that perform a handle time conversion and time processing functions. time_lib is a direct reuse of MABEL code that was ported from the HDF-EOS MDT toolkit (MTDTK5.2.14v1.00).

7.4.1 TAI-UTC Ancillary File

time_lib requires a TAI-UTC ancillary file as input. This file contains leap second information and can be retrieved from the following URL:

ftp://maia.usno.navy.mil/ser7/tai-utc.dat

Updated TAI-UTC files are suggested periodically and required if a leap second has passed since the last update.

7.4.2 Globals

time_lib global variables are listed in Table 7-8.

Table 7-8 time_lib Globals

Module	Variable	Description
const_time_mod	GPS_TO_TAI93	Difference between GPS and TAI epochs (1980-01-06 to 1993-01-01).
const_time_mod	JD_EPOCH_DAY	TAI Julian day of 0 hrs UTC 1-1-93 (whole).
const_time_mod	JD_EPOCH_FRACTION	TAI Julian day of 0 hrs UTC 1-1-93 (fractional).
const_time_mod	JD_1961JAN1	Leap Second-relevant JD.
const_time_mod	JD_1972JAN1	Leap Second-relevant JD.
const_time_mod	MJD_OFFSET	MJD / Julian Date Offset.
const_time_mod	SEC_PER_WEEK	Seconds in a nominal week.
const_time_mod	SEC_PER_DAY	Seconds in a nominal day.

Module	Variable	Description
const_time_mod	SEC_PER_HOUR	Seconds in a nominal hour.
const_time_mod	SEC_PER_MINUTE	Seconds in a nominal minute.
const_time_mod	CCSDS_A_IN_FMT	Format for CCSDS-A timecode (input). YYYY-MM-DDThh:mm:ss.ddddddZ
const_time_mod	CCSDS_A_FMT	Format for CCSDS-A timecode (output). YYYY-MM-DDThh:mm:ss.ddddddZ
const_time_mod	CCSDS_F_FMT	Format for CCSDS-A filename segment. YYYY-MM-DDThhmmss
const_time_mod	CCSDS_B_IN_FMT	Format for CCSDS-B timecode (input). YYY-DDDThh:mm:ss.ddddddZ
const_time_mod	CCSDS_B_FMT	Format for CCSDS-B timecode (output). YYY-DDDThh:mm:ss.ddddddZ
const_time_mod	EOM_DAY	Day number at the end of each month.
const_time_mod	MONTH_DAYS	Max days in each month
const_time_mod	LEAP_OK	Leap Second Status Flags
const_time_mod	LEAP_SEC_IGNORED	Leap Second Status Flags
const_time_mod	NO_LEAP_SECS	Leap Second Status Flags
const_time_mod	ZERO_LEAP_SEC	Leap Second Status Flags
leapsec_mod	g_taiutc	TAI-UTC ancillary data read into core.
leapsec_mod	fs_in_taiutc	fstruct for TAI-UTC ancillary file.

7.4.3 Subroutines

time_lib subroutines are listed in Table 7-9.

Table 7-9 time_lib Subroutines

Module	Subroutine	Description
leapsec_mod	read_taiutc	Reads the TAI-UTC file into a global data structure.
leapsec_mod	get_leapsec	Returns the number of leap seconds for a given JD.
leapsec_mod	print_taiutc	Prints the TAI-UTC data structure.
leapsec_mod	find_taiutc	Sets the filename and path of the ancillary TAI-UTC file.
leapsec_mod	parse_taiutc_cntl	Parses taiutc-related info from the control file.
parse_ccsds_mod	parse_ccsds	Initializes a keyval variable.
timeconv_mod	utc_ccsa_to_ccsb	Converts UTC Time in CCSDS ASCII Time Code A to CCSDS ASCII Time Code B.
timeconv_mod	utc_ccsa_to_file	Converts UTC Time in CCSDS ASCII Time Code A to compressed filename time code.

Module	Subroutine	Description
timeconv_mod	utc_ccsb_to_ccsa	Converts UTC Time in CCSDS ASCII Time Code B to CCSDS ASCII Time Code A
timeconv_mod	jday_to_calday	Converts from integer Julian Day to Calendar Day.
timeconv_mod	gps_s_to_utc_a	Converts from GPS seconds to UTC ASCII.
timeconv_mod	calday_to_jday	Converts from Calendar day to integer Julian Day.
timeconv_mod	jd_to_jdsplit	Converts Julian Date to Toolkit Julian Date Format.
timeconv_mod	mjd_to_jd	Converts Modified Julian Date To Julian Date.
timeconv_mod	jd_to_mjd	Converts from Julian Date to Modified Julian Date.
timeconv_mod	tai93_jd_to_tai93_s	Converts TAI Julian date to time in TAI seconds since 12 AM UTC 1-1-1993. TAI93 is internal toolkit time.
timeconv_mod	tai93_jd_to_utc_jd	Converts from TAI93 JD to UTC JD.
timeconv_mod	tai93_s_to_tai93_jd	Converts from TAI93 seconds to TAI93 JD.
timeconv_mod	tai93_s_to_utc_a	Converts from TAI93 seconds to UTC ASCII (CCSDS-A)
timeconv_mod	tai93_s_to_utc_jd	Converts from UTC JD to UTC ASCII.
timeconv_mod	utc_jd_to_tai93_jd	Converts UTC as a Julian date to TAI as a Julian date.
timeconv_mod	utc_jd_to_utc_a	Converts from UTC JD to UTC CCSDS ASCII.
timeconv_mod	utc_a_to_gps_s	Converts from UTC CCSDS ASCII to GPS seconds.
timeconv_mod	utc_a_to_tai93_s	Converts from UTC CCSDS ASCII to TAI93 seconds.
timeconv_mod	utc_a_to_tai93_jd	Converts from UTC CCSDS ASCII to TAI93 JD.
timeconv_mod	utc_a_to_utc_jd	Converts from UTC CCSDS ASCII to UTC JD.
timeconv_mod	gps_s_to_utc_s	Converts from GPS seconds to UTC J2000 seconds.
timeconv_mod	utc_s_to_utc_a	Converts from UTC seconds to UTC ASCII.
timeconv_mod	mmddyy_to_utca	Converts from MM-DD-YY to UTC ASCII.
timenow_mod	timenow	Returns the system time in CCSDS-A Format.

7.5 math_lib

math_lib contains several Fortran modules that implement mathematical routines. math_lib is a direct reuse of MABEL/GSAS code, with the addition of a polynomial interpolation routine coded for GLAS_HDF.

7.5.1 Globals

math_lib global variables are listed in Table 7-10.

Table 7-10 math_lib Globals

Module	Variable	Description
onepass_avg_mod	onePass_accumulate_TYPE	Statistical type definition. Provides data structure to perform one-pass statistical analysis.

7.5.2 Subroutines

math_lib subroutines are listed in Table 7-11.

Table 7-11 math_lib Subroutines

Module	Subroutine	Description
bilin_interp_mod	bilin_interp	This subroutine calculates the value of properties at a point by doing a bilinear interpolation of the 4 points straddling it.
linearreg_mod	linearreg	Performs linear regression on input array. Returns statistics.
median_mod	print_start_banner	Returns median of a double precision array.
onepass_avg_mod	Onepass_Assign	Assigns elements of one onePass_accumulate_TYPE to another
onepass_avg_mod	Onepass_Init	onePass_accumulate_TYPE
onepass_avg_mod	onePass_Accumulate	Accumulates data for onePass_accumulate_TYPE
onepass_avg_mod	onePass_Compute	Computes the min, mean, max and standard deviation for onePass_accumulate_TYPE
onepass_avg_mod	Onepass_Print	Prints onePass_accumulate_TYPE

7.6 cntl_lib

cntl_lib contains several Fortran modules that handle control file processing. cntl_lib is a direct reuse of MABEL/GSAS code.

7.6.1 Control Files

Control files provide dynamic control information to PGEs. Most PGEs are designed to take the name of the control file as a command-line argument during each invocation of the PGE. Most PGEs should terminate with a fatal error if the command-line argument is missing, the specified file does not exist, or the file is unreadable.

Control files are designed to be part of a larger control file used by one or more PGEs. The larger control file includes sections that identify the PGE that will perform the task requiring the inputs contained in the section. Each section is bounded by an "=" sign in column 1, followed by the PGE name that requires the control inputs.

All control files are created in standard “keyword=value” format. This format is text-based and consists of a line containing a keyword/value pair delimited by an equal sign (=). The ordering of the keywords is not relevant but should follow a convention for consistency.

Multiple instances of certain keywords are allowed. The keyword is not case sensitive. Spaces are allowed, but not required. Comment lines must be prepended by a "#" character. The keyword is limited to MAXSTR characters; the value is limited to MAXLINE characters.

Control file examples and allowable values are defined in the detailed design section for each GLAS_HDF PGE.

7.6.2 Subroutines

cntl_lib subroutines are listed in Table 7-11.

Table 7-12 math_lib Subroutines

Module	Subroutine	Description
cntl_mod	get_secstart	Positions file pointer to start of the section with a control file.
cntl_mod	open_cf	Opens the control file for reading.
cntl_mod	read_cf	Reads the control file into a keyval structure.
cntl_mod	close_cf	Closes the control file.
cntl_mod	print_cf	Prints the control file.
cntl_mod	check_cntl	Checks control to see if there are unused control lines.
filecntl_mod	parse_filecntl	Parses control structures for input/output files.
filecntl_mod	parse_fileinfo	Parses file control structures for file parameters.
filecntl_mod	read_filendx	Reads a file index (control-style list of files).
globcntl_mod	parse_globcntl	Parses control settings for common_lib routines.
globcntl_mod	parse_err_cntl	Parses error-related info from the control file.

7.7 anc_lib

anc_lib contains several Fortran modules that handle ancillary data files. anc_lib is a direct reuse of MABEL/GSAS code. Since none of the anc_lib code is used in GLAS_HDF, the library will not be documented here.

7.8 hdf_lib

hdf_lib contains several Fortran modules that provide an interface to the HDF5_LIBRARY for handling HDF5 files in the MABEL/GLAS_HDF style. hdf_lib was created for MABEL and has been improved by the GLAS_HDF effort. These improvements will be rolled back into the MABEL codebase.

hdf_lib provides routines that implement conventions of the GLAS_HDF file format. The constructs that implement these conventions map directly to several GLAS_HDF requirements and include:

- Parameters implemented as HDF5 chunked/compressed datasets.

- CF parameter attributes
- CF global attributes
- Descriptive labeling

h5_param_mod and h5_param2_mod are two nearly identical module implementations. The only difference is that h5_param_mod handles parameters of rank=1 whereas h5_param_mod handles parameters of rank=2. As such, only h5_param_mod routines will be documented.

hdf_lib also contains the h5_codegen_module. This module provides generic routines for the code generator.

7.8.1 Parameters

Parameters are written to a HDF5 file as chunked/compressed datasets. There are two major programming components used to instantiate parameters and several subroutines used to read and write parameters in a consistent manner. hdf_lib currently only provides support for one and two-dimensional datasets.

h5_param_type1 (and h5_param_type2) are type definitions that contain all the information to instantiate parameters needed by the HDF5_LIBRARY. These type definitions contain no actual scientific data values, but contain elements needed to read and write the scientific data values. Components are described in Table 7-13.

Table 7-13 h5_param_type

Component	Description
label	HDF5 identifying label
m_dtype	HDF5 Data type as represented in memory
f_dtype	HDF5 Data type a stored within a HDF5 file.
did	HDF5 dataset id
sid	HDF5 dataspace id
pid	HDF5 plist id
mid	HDF5 memoryspace id
fid	HDF5 filespace id
rank	Rank of data.
gid	HDF5 group ID were the parameter would be read/written.
start	Start position in the data array for read/write.
size	Total number of elements read/written.
max_dims	Total size of data array on disk.
dims	Size of each chunk.
slen	Length of string for character data.

Component	Description
zip_lvl	Zip compression level (for SHUFFLE/GZIP compression)
precision	Precision saved to disk (For scale/offset compression – not used in GLAS_HDF)
att	Data structure of descriptive attributes that will be attached to the parameter.

7.8.2 CF Parameter Attributes

Each parameter written to a HDF5 file includes attached CF attributes that describe the parameter and provide information both on the HDF5 file and for the generated data dictionary.

h5_pattr_type is a type definition that contains elements which instantiate the CF parameter attributes. Each element is a text string that will be internally converted to an appropriate datatype (if applicable) when written to the HDF5 file. If an attribute contains the string "not_set", the attribute is not written to the HDF5 file. Table 7-14 describes each element and (if applicable) its corresponding CF attribute name.

Table 7-14 Parameter Attributes

Attribute	Description (CF indicates a standard CF attribute.)
name	Descriptive name of the parameter. (CF : long name)
standard_name	A standard name that references a description of a variable's content in the standard name table. (CF : standard_name)
units	Units of a variable's content. (CF : units, compliant with NetCDF UDUNITS)
hertz	Data rate of the parameter (occurrences per second).
description	Description of the parameter.
source	Method of production of the original data. (CF: source)
coordinates	Identifies auxiliary coordinate variables, label variables, and alternate coordinate variables. (CF : coordinates)
valid_min	Smallest valid value of a variable. (CF: valid_min)
valid_max	Largest valid value of a variable. (CF: valid_max)
flag_values	Provides a list of the flag values. Use in conjunction with flag_meanings. (CF : flag_values)
flag_meanings	Use in conjunction with flag_values to provide descriptive words or phrases for each flag value. If multi-word phrases are used to describe the flag values, then the words within a phrase should be connected with underscores. (CF : flag_meanings)
fillvalue	A value used to represent missing or undefined data. Not allowed for coordinate data except in the case of auxiliary coordinate variables in discrete sampling geometries. (CF : _FillValue)

Parameter CF attribute descriptions were copied from :

http://cf-pcmdi.llnl.gov/documents/cf-conventions/1.6/cf-conventions.html.

7.8.3 CF Global Attributes

CF global attributes are the primary source of human-readable metadata on GLAS_HDF products. The attributes are a subset of relevant EOSDIS ECHO metadata fields merged with a subset of relevant CF attribute fields. The global attributes are attached to the root level of the HDF5 file.

h5_file_cf is a global keyval structure that contains elements which instantiate the CF global attributes. If an attribute contains the string "not_set", the attribute is not written to the HDF5 file. Table 7-15 lists the supported CF global attributes.

Table 7-15 CF Global Attributes

Attribute	Description
accessconstraints	Describes access constrains imposed upon the data by the producer. (ECHO)
campaign	GLAS_HDF specific campaign identifier.
citationforexternalpublication	Describes the citation required when citing the data in a formal publication. (ECHO)
comment	Miscellaneous information about the data that cannot be described in any of the other available attributes. (CF)
contributor_name	The name of any individuals or institutions that contributed to the creation of this data. Listed contributors must be comma separated and same order as listed in the contributor_role attribute. (CF)
contributor_role	The role of the individual or institution that contributed to the creation of this data. Listed roles must be comma separated and in the same order as listed in the contributor_name attribute. (CF)
Conventions	States that the CF convention is being used and what version. (CF)
creator_email	Email address of the person/organization that created the data. (CF)
creator_name	Name of the person/organization who created the data. (CF)
date_created	The date or date and time when the file was created. (CF)
date_type	Time epoch under which timestamps are represented (ECHO).
featureType	A featureType describes the fundamental relationships among the spatiotemporal coordinates (CF).
geospatial_lat_max	Maximum latitude coordinates of the bounding box of the data set. (CF)
geospatial_lat_min	Minimum latitude coordinates of the bounding box of the data set. (CF)
geospatial_lat_units	Defines the units applied to the geospatial_lat_min and geospatial_lat_max attributes. (CF)
geospatial_lon_max	Maximum longitude coordinates of the bounding box of the data set. (CF)
geospatial_lon_min	Minimum longitude coordinates of the bounding box of the data set. (CF)

Attribute	Description
geospatial_lon_units	Defines the units applied to the geospatial_lon_min and geospatial_lon_max attributes. (CF)
hdfversion	Version of HDF5_LIBRARY used to product the data.
history	List of any changes made to the file. (CF)
identifier_file_uuid	Machine readable unique identifier for each file. (ECHO)
identifier_product_doi	Machine readable digital object identifier (DOI) for each product type. (ECHO)
identifier_product_doi_authority	Authority where values for indentifier_product_DOI are registered. (ECHO)
institution	The institution of the person or group that collected the data.
instrument	Name of the instrument that was used in the collection of the data. (CF)
keywords	A comma separated list of Global Change Master Directory (GCMD) key words and phrases. (CF/ECHO)
keywords_vocabulary	Identifies the controlled list of keywords from which the values in the "keywords" attribute are taken. (CF)
license	Describes the restrictions to data access and distribution. (CF)
platform	Name of the platform that was used in the collection of the data. (CF)
processing_level	EOSDIS identifier describing the processing level of the data (CF/ECHO)
project	The scientific project that the data was collected under. (CF)
publisher_email	The email address of the person/organization that distributes the data files. (CF)
publisher_name	Name of the person/organization that distributes the data files. (CF)
publisher_url	URL of the person/organization that distributes the data files. (CF)
references	Contains published or web-based references that describe the data or methods used to produce it.
shortname	EOSDIS identifier indicating product type. (ECHO)
source	The method of production of the original data. (CF)
spatial_coverage_type	Spatial coverage type of the data. (ECHO)
standard_vocabulary_name	The name of the controlled vocabulary from which variable standard names are taken. (CF)
summary	One paragraph describing the data set. (CF)
time_coverage_duration	Describes the temporal coverage duration of the data (CF).
time_coverage_end	Describes the temporal coverage end of the data (CF).
time_coverage_start	Describes the temporal coverage start of the data (CF).
time_type	Time standard under which timestamps are represented (ECHO).

Attribute	Description
title	Short description of the data contained within the file. (CF)

Some CF Global attribute descriptions were copied from :

http://www.nodc.noaa.gov/data/formats/netcdf/#guidencetable

7.8.4 Descriptive Labeling

The descriptive labeling requirement is mostly fulfilled by the CF parameter and global attributes conventions. The unmet implied requirement remaining is to provide a description for each HDF5 group created. The h5_create_group subroutine requires a description as an argument when creating a new group.

7.8.5 Globals

hdf_lib global variables are listed in Table 7-10.

Table 7-16 hdf_lib Globals

Module	Variable	Description
cf_attr_mod	h5_file_cf	Keyval list of cf-style metadata attributes that will be attached to the root level.
h5_param_mod	h5_param_type1	h5_param type definition. The h5_param structure contains all required HDF5 internal information about a parameter.

7.8.6 Subroutines

hdf_lib subroutines are listed in Table 7-11.

Table 7-17 hdf_lib Subroutines

Module	Subroutine	Description
cf_attr_mod	h5_write_file_cf	Attaches the list of h5_file_cf attributes to the root group of a HDF5 file.
cf_attr_mod	h5_read_file_cf	Reads h5_file_cf attributes from the root group of a HDF5 file.
cf_attr_mod	write_file_cf_data_dict	Writes h5_file_cf attributes in HTML-based data dictionary format.
h5_attr_mod	h5_write_pattr	Attaches a set of CF-style parameter attributes to a HDF5 parameter.
h5_attr_mod	h5_read_pattr	Reads CF parameter attributes from a HDF5 parameter.
h5_attr_mod	h5_init_pattr	Initializes a set of CF parameter attributes.
h5_attr_mod	h5_print_pattr	Prints CF parameter attributes in HTML-based data

Module	Subroutine	Description
		dictionary format.
h5_attr_mod	h5_print_pattr_head	Prints a header for CF parameter attributes in HTML-based data dictionary format.
h5_attr_mod	h5_set_r8_attr	Sets CF min/max/fill values for an r8 parameter.
h5_attr_mod	h5_set_r4_attr	Sets CF min/max/fill values for an r4 parameter.
h5_attr_mod	h5_set_i4_attr	Sets CF min/max/fill values for an i4 parameter.
h5_codegen_mod	write_module_start	Writes code for the module start & typedefs.
h5_codegen_mod	write_init	Write code for the init routine.
h5_codegen_mod	write_alloc	Writes code for the alloc/dealloc routines.
h5_codegen_mod	write_open	Writes code for the group open routine.
h5_codegen_mod	write_close	Writes code for the group close routine.
h5_codegen_mod	write_h5init	Writes code for the group H5 init routine.
h5_codegen_mod	write_create	Writes code for the group create routine.
h5_codegen_mod	write_read	Writes code for the group read routine.
h5_codegen_mod	write_write	Writes code for the group write routine.
h5_codegen_mod	write_print_attr	Writes code for the attributes print routine.
h5_codegen_mod	write_print_head	Writes code for the header print routine.
h5_codegen_mod	write_print_data	Writes code for the data print routine.
h5_codegen_mod	write_data_dict	Writes code for the data dictionary routine
h5_codegen_mod	write_sync	Writes code for the data sync routine.
h5_codegen_mod	write_setds	Writes code for the dimension scale routine.
h5_codegen_mod	write_end	Writes code for the module end.
h5_file_mod	openr_h5_file	Opens a HDF5 file for input.
h5_file_mod	openw_h5_file	Creates a HDF5 file for output.
h5_file_mod	close_h5_file	Closes a HDF5 file.
h5_group_mod	h5_create_group	Creates a HDF5 group with a description attribute and custom property list settings.
h5_param_mod	h5_create_param	Creates a parameter as a chunked dataset; writes parameter attributes.
h5_param_mod	h5_open_param	Opens a parameter as a chunked dataset; reads parameters attributes.
h5_param_mod	h5_close_param	Closes constructs associated with a parameter chunked dataset.
h5_param_mod	h5_extend_param	Extends a dataset for writing a parameter as a chunked dataset.

Module	Subroutine	Description
h5_param_mod	h5_select_chunk	Selects a hyperslab for reading a parameter as a chunked dataset.
h5_param_mod	h5_read_i_param	Reads a parameter as an integer chunked dataset.
h5_param_mod	h5_read_f_param	Reads a parameter as a floating-point chunked dataset.
h5_param_mod	h5_read_d_param	Reads a parameter as a double precision chunked dataset.
h5_param_mod	h5_read_s_param	Reads a parameter as a string chunked dataset.
h5_param_mod	h5_write_i_param	Writes a parameter as an integer chunked dataset.
h5_param_mod	h5_write_f_param	Writes a parameter as a floating-point chunked dataset.
h5_param_mod	h5_write_d_param	Writes a parameter as a double precision chunked dataset.
h5_param_mod	h5_write_s_param	Writes a parameter as a string chunked dataset.
h5_param_mod	h5_init_param	Initializes a parameter as a chunked dataset.
h5_param_mod	h5_print_pinfo	Prints parameter information in data-dictionary style HTML format.
h5_param_mod	h5_print_pinfo_head	Prints a header for parameter information in data-dictionary style HTML format.
h5_param_mod	h5_print_pinfo_end	Prints HTML to close a table created by h5_print_pinfo_head.
h5_param_mod	h5_quickinit_param	Initializes a parameter using passed arguments.

7.9 Dependencies

Library code is implemented in separate directories and grouped by functional area. A single Makefile in each library subdirectory will compile the subdirectory source code into a statically-linked library. A cascading Makefile at the top level of the common_libs source tree will compile all the libraries in one step.

A hierarchy of dependencies exist between the libraries. The order in which libraries are compiled (and linked) is important since libraries may depend upon other libraries for support routines. This is not an issue if the developer uses the supplied Makefile infrastructure, but the developer should be aware that these dependencies exist. The dependency structure is illustrated in Table 7-18.

Table 7-18 common_lib Libraries and Dependencies

To build...	The following libraries are required...
const_lib	<none>
err_lib	const_lib
mutil_lib	const_lib, err_lib
time_lib	const_lib, mutil_lib

math_lib	const_lib, err_lib
cntl_lib	const_lib, err_lib, mutil_lib
anc_lib	mutil_lib, math_lib
hdf_lib	const_lib, err_lib, HDF5_LIBRARY

8.0 GSAS_LIB/GLASHDF_LIB IMPLEMENTATION NOTE

Whereas the GLAS_HDF architecture and design documentation represents gsas_lib and glashdf_lib as two separate entities, they have been implemented in a single static library. This decision was made because the only unique functionality provided within gsas_lib is implemented within a two modules that handle the GLAS_HDF-specific implementation of metadata.

Additionally, the product-specific routines discussed within gsas_lib documentation have been implemented as subroutines directly callable within each product-specific PGE. This decision was made to enable the team to deliver individual product-conversion PGEs without having to re-deliver the static libraries.

9.0 GSAS_LIB

gsas_lib is a collection of GSAS routines re-used to read the GLAS_BIN products. Primary functions provided by these routines include:

- Reading GLAS_BIN product headers (metadata)
- Reading and converting GLAS_BIN data into scientific units
- Unpacking GLAS_BIN bit flags
- Handling GLAS_BIN QA data
- Handling GLAS_BIN PASSID/NOSE information
- Writing MET detached metadata files.

9.1 Support Modules

Routines within this library that are non-product-specific are GSAS legacy support routines required by the product-specific routines. Functionality provided by each support module is listed in Table 9-1. Due to their limited usefulness, individual subroutines are not documented here.

Table 9-1 GSAS_LIB Support Modules

Module	Description
MetaQA_mod	Parses QA metadata information passed via control
WriteMetaFile_mod	Writes metadata information to a detached metadata (MET) file in ESDIS-compliant ECHO format.
anc45_meta_mod	(required only for support)
c_compare_mod	(required only for support)
c_nose_mod	Handles NOSE information required within detached metadata files.
common_flags_mod	Contains routines for packing/unpacking flags that appear in multiple GLAS_BIN products.
common_hdr_mod	Provides data structures and routines for manipulating GLAS_BIN product headers.
const_gsas_mod	Provides GSAS-specific constants.
conversions_mod	Provides generic datatype conversions.
get_numhdrs_mod	Returns the number of header records within any GLAS_BIN product.
kinds_mod	Provides GSAS data type definitions.
passid_mod	Handles passid information passed by control and required for NOSE support.
prod_def_mod	Defines the record sizes of each GLAS_BIN product.

9.2 Product-Specific Modules

Product-specific modules constitute an API for each GLAS_BIN product type. These modules were designed and implemented to provide a standard set of functions for manipulating each product type. It will be sufficient to abstractly describe a model API designated as “glaxx”.

The GLAS_BIN APIs are instantiated as a set of at most six modules. If a product contains no product-specific flags, the flag module (glaxx_flags_mod) does not exist. If a product contains no product-specific metadata, the header module (glaxx_hdr_mod) does not exit. The primary use of the GLAS_BIN API within GLAS_HDF is to read the product records and convert the integer product variables into scientific units.

Table 9-2 Product-Specific Modules

Module	Description
glaxx_prod_mod	Defines product-specific record format and associated global product data structure. Each module also includes one subroutine to initialize the product data and another to print the data in a human-readable form.
glaxx_hdr_mod	Contains routines to read and write product-specific metadata information.
glaxx_alg_mod	Defines product-specific global algorithm (scientific units) data structure. Each module also includes one subroutine to initialize the algorithm data and another to print the data in a human-readable form
glaxx_scal_mod	Defines product-specific global scaling data structure. Also includes subroutines to initialize the scaling data, convert from product to algorithm format (GLAxx_P2A), convert from algorithm to product format (GLAxx_A2P), and print the scaling data in a human-readable form.
glaxx_flag_mod	Contains routines for packing/unpacking product-specific flags.

10.0 GLASHDF_LIB COMPONENT

glashdf_lib contains functionality that is specific to the GLAS_HDF effort. Since most of this GLAS_HDF functionality existed within the reused common_libs and gsas_lib, glashdf_lib provides only functions for handling GLAS_HDF metadata.

10.1 GLAS_HDF Metadata

GLAS_HDF metadata is derived from a combination of ECHO-style inventory-level metadata present within GLAS_BIN product headers, collection and inventory-level metadata provided by ECS-generated ESDT descriptor files, information provided via control, and some additional metadata created on the fly. The input metadata is merged into a global metadata construct that used to fill four distinct flavors of metadata on the GLAS_HDF product: ancillary_data, provenance metadata, grouped metadata and global metadata.

Global metadata is implemented as attributes attached to the root level of the GLAS_HDF file. Provenance metadata contains information about the processing history of the GLAS_HDF file. Most of the metadata contained within the GLAS_BIN product headers is also contained with the ESDT file. Any metadata item within the GLAS_BIN headers that is also present within the ESDT file is written as grouped metadata on the product. However, some GLAS_BIN products do contain metadata that is not within the ESDT file. In this case, those metadata are written to the ancillary_data group on the product.

The GLAS_HDF metadata flow is shown in Figure 10-1.

10.1.1 Global Metadata

GLAS_HDF global metadata routines are instantiated by common_libs/hdf_lib. The routines within glashdf_lib simply copy appropriate values contained with the merged metadata structure to the h5_file_cf structure. Routines within hdf_lib are called to write the metadata to the product. Please refer to the hdf_lib section for detailed information regarding global metadata.

10.1.2 Grouped Metadata

Grouped Metadata contains the filled content of the EOSDIS ESDT Metadata Configuration File (MCF). The MCF describes metadata values that will be ingested into the ECS databases. In order to store the MCF information as attributes attached to HDF5 groups attributes, some reorganization and re-labeling was necessary. However, all of the information described within the MCF file is present within the grouped metadata. The benefit of the grouped metadata is that it is easily computer-parseable and enables the mechanism by which .MET metadata ingest files are created for the EOSDIS datacenter.

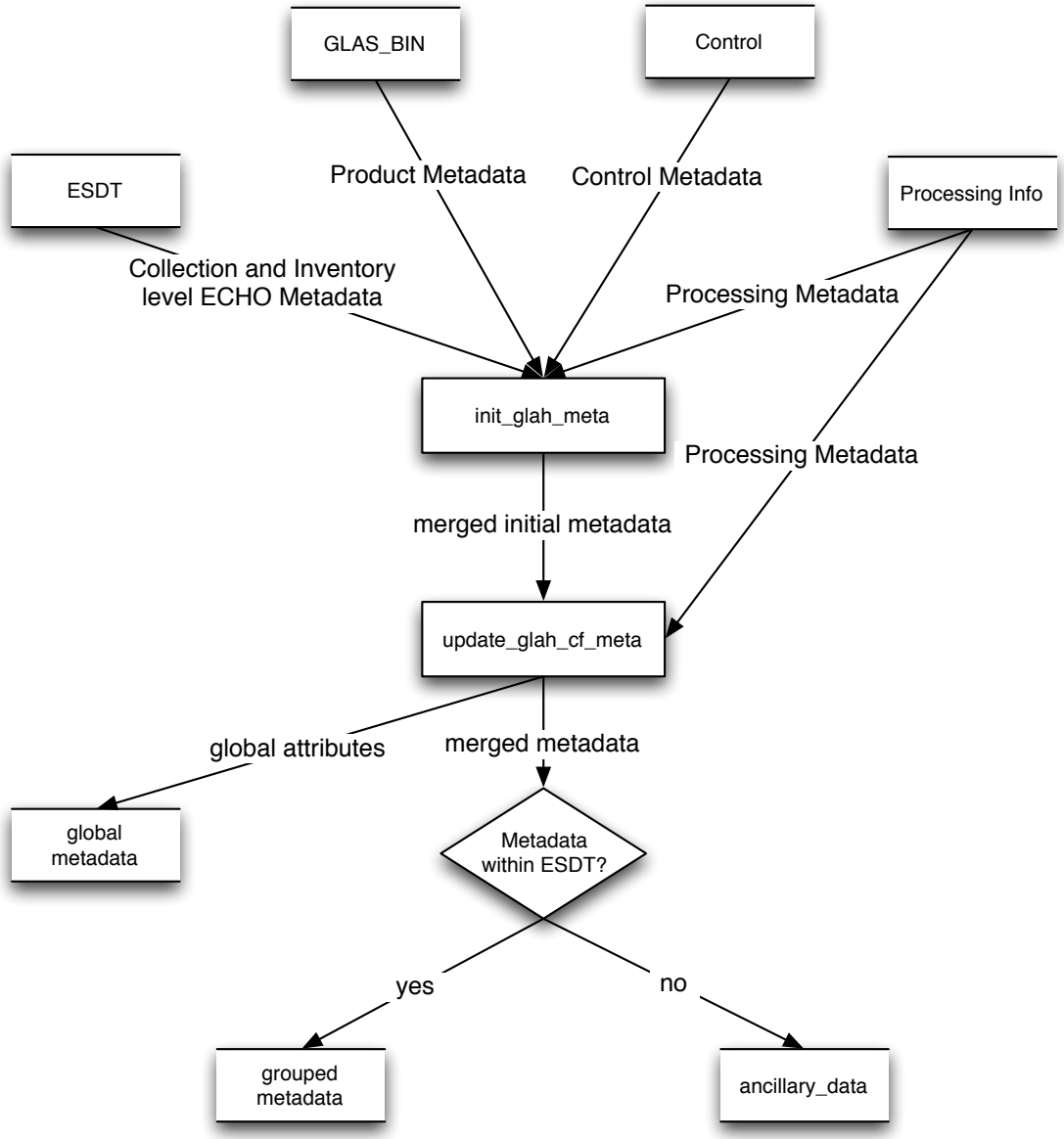

Figure 10-1 GLAS_HDF Metadata Flow

10.1.3 ancillary_data

The ancillary_data group contains metadata information present on certain GLAS_BIN product types that is not also present in the ESDTs. This is also the place where any additional metadata or ancillary information can be stored.

10.1.4 Provenance Metadata

The process that converts GLAS_HDF data from an integer-binary format into HDF5 is transformative. However, there is a requirement to keep provenance information

regarding the process that created the original GLAS_BIN file since that contains important traceability information.

Without a conclusive existing standard to define provenance, the GLAS_HDF provenance implementation is focused on instrumenting the product with the information necessary to generate a provenance map via external software. The goal was to provide enough information, in an identifiable fashion, that external software can generate a provenance map in the format of its choosing.

Provenance metadata is stored on the products within a provenance group as a series of numbered step groups representing a step in the processing history.

Example step group: /METADATA/PROVENANCE/STEP_1

Each step group contains the following attributes:

Attribute	Description
ProcessDateTime	Date/time of processing step completion.
ProcessAgent /Name	Name of processing software.
ProcessAgent /Type	Type of processing software.
ProcessAgent /Version	Version of processing software.
ProcessAgent /Description	Description of processing performed.
ProcessInput /Name	Comma-separated list of processing input files.
ProcessInput/Type	Comma-separated list of processing input file types. (same order above)
ProcessInput /Version	Comma-separated list of processing input file versions. (same order above)
ProcessOutput /Name	Comma-separated list of processing output files.
ProcessOutput /Type	Comma-separated list of processing output file types. (same order above)
ProcessOutput /Version	Comma-separated list of processing output file versions. (same order above)
ProcessOutput /UUID	Comma-separated list of processing output file UUIDs. (same order above)
ProcessOutput /DOI	Comma-separated list of processing output file DOIs. (same order above)

In addition, GLAS_HDF was part of an EOSDIS pilot project to instrument earth science granules with DOIs (digital object identifiers). Each GLAS_HDF product type has a unique DOI registered with the International DOI Foundation (http://www.doi.org). A DOI can be used, for example, to uniquely identify each datatype cited in a research paper.

Expanding upon the EOSDIS DOIs, each GLAS_HDF granule will also been assigned a Universally Unique Identifier (UUID) that can be used to uniquely identify each individual GLAS_HDF granule.

The DOI and DOI authority values are contained within the ESDT files. The UUID is passed via control. All values are stored in both the global metadata and grouped metadata structures.

10.1.5 Globals

Selected glashdf_lib global variables are listed in Table 10-1. Other global variables contain names of metadata fields and control keywords. These are too numerous to document here.

Table 10-1 glashdf_lib Globals

Module	Variable	Description
glah_meta_mod	esdt_label	Parsed ESDT labels
glah_meta_mod	esdt_param	Parsed ESDT parameter
glah_meta_mod	glas_meta	Metadata from GLAS Headers
glah_meta_mod	glah_label	Merged metadata labels
glah_meta_mod	glah_param	Merged metadata parameters

10.1.6 Subroutines

glashdf_lib subroutines are listed in Table 10-2 .

Table 10-2 glashdf_lib Subroutines

Module	Subroutine	Description
glah_meta_mod	fix_container	Replaces ESDT containers with the name of their content.
glah_meta_mod	parse_meta_cntl	Parses ESDT and metadata-related info from control.
glah_meta_mod	parse_esdt_meta	Parses metadata structures from the ESDT file
glah_meta_mod	init_glah_meta	Merges metadata from the ESDTS, GLAS headers, and additional metadata. It builds a combined set of metadata paths and values.
glah_meta_mod	update_glah_cf_meta	Copies selected structured metadata values to CF global attributes.
glah_meta_mod	write_inv_meta_at	This routine writes metadata as an HDF5 attribute.
glah_meta_mod	h5_write_glah_meta	Writes global and grouped metadata to a HDF5 file.
glah_meta_mod	h5_close_glah_meta	This routine closes any open metadata groups.
glah_meta_mod	print_glah_meta_dd	Prints all metadata in a HTML data dictionary format.
glah_meta_mod	print_glah_meta_group	Prints a metadata group in HTML data dictionary format.
glah_meta_mod	count_glah_meta	Counts the number of attributes in a Metadata group
glah_meta_mod	read_glah_meta	Reads HDF5 Metadata.
parse_gla_meta_mod	parse_gla_meta	Parses metadata from GLAS headers

11.0 GLAHXX_API COMPONENT

The interface to each GLAS_HDF product is implemented as product-specific Fortran modules that contain the routines necessary to read, write and document each product. The API makes extensive use of functionality provided by common_lib, especially hdf_lib. glah_codegen creates the majority of the functionality implemented by the API within its generated code.

11.1 Rate Groups

Since GLAS data are multi-rate (i.e.: some 40Hz, some 1Hz, etc.), the GLAS_HDF products incorporate "rate groups". Rate groups are top-level groups labeled with a data rate containing all parameters of that particular data rate. Each rate group has a time parameter and corresponding latitude/longitude that correspond in a 1-to-1 fashion with other data parameters within that rate group. Each rate group has a description that provides information about the group.

As written by the gla_codegen PGE, there is a separate routine to handle each rate-group (groups of parameters with the same data rate) on the product. Each rate group module contains a science data structure defining the actual science data content of the rate-group, a corresponding structure consisting of h5_param_type substructures that contain corresponding HDF5 parameter information, and an allocable array of science data structures.

The input/output routines are designed to read/write chunked datasets. However, in practice, it is faster to read and write whole datasets. The capability to chunk datasets is still required for gzip compression, but the API hides chunking complexity from a user of the routines.

In the following examples and in the global and subroutine sections, "rg" will be used as an example rate group identifier.

To write a rate group named "rg", subroutines should be called in this order:

```
call h5_create_glahxx_rg
call h5_write_ glahxx_rg _chunk
call h5_set_ glahxx_rg_ds
call h5_close_ glahxx_rg
```

To read a rate group named "rg", subroutines should be called in this order:

```
call h5_open_ glahxx_rg
call h5_read_ glahxx_rg _chunk
call h5_close_ glahxx_rg
```

Note that the glahxx_api routines are not required to read data from a GLAS_HDF file. In many cases, especially when a programmer only wants to read one or a few parameters, it is just as easy to use the H5LT (HDF5 Lite) API provided with the HDF5_LIBRARY.

11.2 Logical Groups

GLAS products have lots of parameters. There are 15 GLAS products containing a total of over 2000 parameters. To bring some order to the parameters, logical groups (within each rate group) are implemented to organize the data by discipline or topic. Each logical group has a description that provides information about the group. Logical grouping of parameters is implemented by glahxx_api.

11.3 Dimension Scales

Dimension scales are the mechanism by which NetCDF associates array dimensions. For complete NetCDF compliance, a dimension scale is needed for every single or multiple dimensioned parameter on the product. Multiple parameters may share the same dimension scale as long as their array lengths are the same (and a shared scale makes sense).

This means that for every parameter "z(x)", there must be a dimension scale "y" that has dimensions equivalent to "z" and has a value corresponding to each element of the z array. Extending this to two dimensions, for every parameter z(x,y), there must be two dimensions scales "x" and "y" with number of elements equal to the respective dimensions of z and containing corresponding values. Additional requirements are that :

- Any parameter identified as a dimension scale must be stored within the same group or within a higher-level group than any other parameter that references it.
- No dimension scale may contain invalid values (or have a _FillValue attribute attached).

A single time-based dimension scale is implemented automatically within glahxx_api. However, since some GLAS_BIN products contain two dimensions, additional dimension scales must be implemented in order to maintain NetCDF compliance. The majority of this code is added to the h5_set_glahxx_rg_ds subroutine

The most important part of this process is deciding what the dimension scale should represent. For example, consider a two-dimension array of backscatter profiles where samples are taken at regular intervals for entire profile within the atmosphere column. The first dimension scale is time since the array is time-based. The second dimension scale corresponds to the range of the sample within the atmosphere column. The second dimension scale could contain several representations. For example:

- An array of integers with values 1-n representing the "index" to the profile measurement.
- An array of floating point values representing the top range of each measurement value for its integration.
- An array of floating point values representing the midpoint of each measurement value for its integration.

Once the representation is determined, the actual coding can begin. In code generated by gla_codegen, there are example commented code fragments which describe how to implement a custom dimension scale.

To understand what NetCDF does with this information, ncdump would describe this parameter as such:

```
cloud_indicator_flags(DS_utctime, DS_range_window_top)
```

By convention, all GLAS_HDF dimension scale labels start with "DS_".

11.4 Globals

glahxx_api global variables are listed in Table 11-1. "rg" will be used as an example rate group identifier.

Table 11-1 glahxx_api Globals

Module	Variable	Description
glaxx_rg_mod	g_h5_glahxx_rg_chunksize	HDF5 chunksize of each HDF5 dataset.
glaxx_rg_mod	glaxx_rg_type	Type definition defining a structure containing the science data content.
glaxx_rg_mod	g_glahxx_rg	Single-instance instantiation of the glaxx_rg_type. (A "record" in past terminology).
glaxx_rg_mod	g_glaxx_rg_buff	Allocable array of glaxx_rg_types (A "data buffer" in past terminology).
glaxx_rg_mod	g_glaxx_rg_dim_gid	Group ID where dimension scales are stored.
glaxx_rg_mod	h5_glaxx_rg_type	Type definition that contains HDF group ids for each logical group and h5_param_type substructures for each HDF5 dataset.
glaxx_rg_mod	g_h5_glahxx_rg	Instantiation of h5_glaxx_rg_type.
glaxx_rg_mod	h5_glahxx_rg_label_type	Type definition defining a structure containing text labels for each logical group and each dataset.
glaxx_rg_mod	g_h5_glahxx_rg_label	Instantiation of h5_glahxx_rg_label_type.
glaxx_rg_mod	h5_glaxx_rg_group_desc_type	Type definition defining a structure containing descriptive information for each group.
glaxx_rg_mod	g_h5_glaxx_rg_group_desc	Instantiation of h5_glaxx_rg_group_desc_type.

11.5 Subroutines

glahxx_api subroutines are listed in Table 11-2. "rg" will be used as an example rate group identifier.

Table 11-2 glahxx_api Subroutines

Subroutine	Description
init_glahxx_rg	Return an initialized glahxx_rg structure.
h5_init_glahxx_rg	Initializes the HDF5 data and attributes within the g_h5_glahxx_rg structure and allocates the g_glaxx_rg_buff buffer.
allocate_glahxx_rg	Allocates the g_glaxx_rg_buff buffer.
deallocate_glahxx_rg	Deallocates the g_glaxx_rg_buff buffer.
h5_create_glahxx_rg	Initializes the rate group for writing – creates the rate group, logical groups, and HDF5 datasets.
h5_open_glahxx_rg	Opens the rate group for reading – opens the rate group, logical groups, initializes the g_h5_glahxx_rg structure and opens the HDF5 datasets.
h5_close_glahxx_rg	Closes the HDF5 datasets, logical groups and the rate group.
h5_read_glahxx_rg_chunk	Reads a chunk of science data into the g_glaxx_rg_buff buffer.
h5_write_glahxx_rg_chunk	Writes the chunk of data within the g_glaxx_rg_buff buffer to the HDF5 file.
print_glahxx_rg_attr	Prints the groups and attributes of a rate group in TAB-delimited text format.
print_glahxx_rg_head	Prints headers corresponding to print_glahxx_rg_data output in a TAB-delimited text format.
print_glahxx_rg_data	Prints dataset values in a TAB-delimited text format.
print_glahxx_rg_data_dict	Prints the rate group data dictionary in HTML format.
sync_glahxx_rg	Returns the buffer index value of data corresponding to a particular time.
h5_set_glahxx_rg_ds	Converts datasets to dimension scales and links the dimension scale to other dataset.

12.0 GLAXX_H5_CONVERT COMPONENT

glaxx_h5_convert is the model of a PGE that transforms a GLAS_BIN file into GLAS_HDF. An instance of this model exists for each GLAS_HDF product type. glaxx_h5_convert leverages all the software previously described in this document.

gla_codegen creates a significant amount of the code necessary to instantiate glaxx_h5_convert, but some code must be added by the programmer to handle copying multi-rate data from the GLAS_BIN to the GLAS_HDF data structure.

gla_codegen consists of two major items: a main Fortran program (glaxx_h5_convert) and an initialization subroutine (main_init).

12.1 main_init

main_init provides initialization functions for glaxx_h5_convert. The functions include (in order):

- Read and parse the control file.
- Open and read the requisite TAI-UTC file.
- Parse the control file for control overrides.
- Initialize the GLAS_BIN scale factors.
- Parse file input/out information from the control file.
- Open the input GLAS_BIN file.
- Open and parse the requisite corresponding ESDT file.
- Create the output GLAS_HDF file.
- Initialize the metadata.
- Verify all control file entries were parsed.

Most of this functionality is incorporated within calls to library routines. Figure 12-1 shows this graphically.

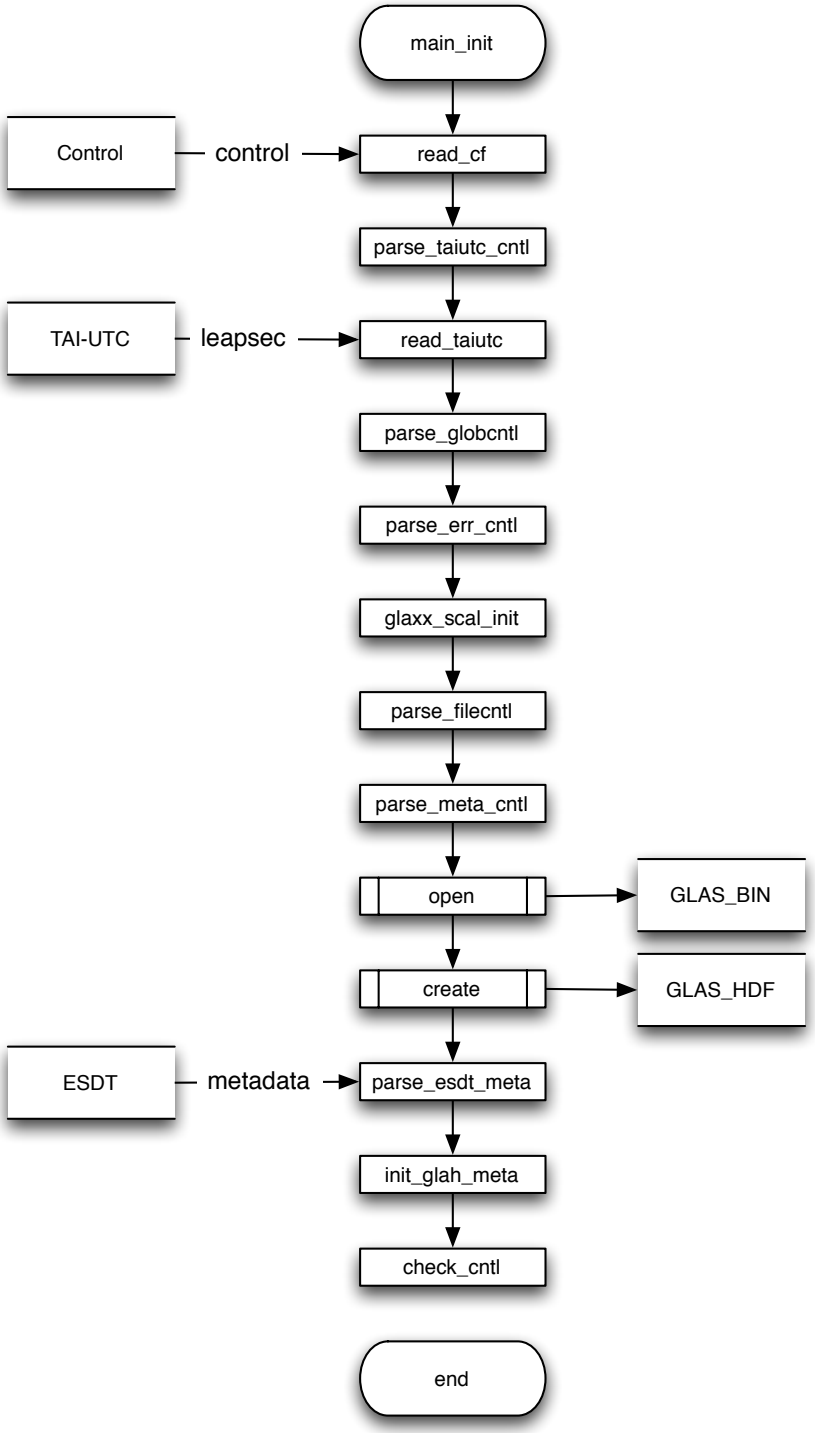

Figure 12-1 main_init

12.2 glaxx_h5_convert

glaxx_h5_convert is the Fortran program that implements the PGE. It is mostly a processing shell that calls library routines and the glahxx_api subroutines. The primary unique functionality implemented in glaxx_h5_convert involves copying data from the GLAS_BIN to the GLAS_HDF data structures. Execution flows through glaxx_h5_convert as follows:

- Initialize global constants
- Print status information
- Open the HDF5_LIBRARY
- Call main_init
- Initialize and allocate each rate group
- For each record on the GLAS_BIN product…
    - Read the record
    - Convert the GLAS_BIN product variables to algorithm (scientific units) variables.
    - Copy data from the GLAS_BIN data structure to the GLAS_HDF data structure, performing any required interpolation. Skip any spare or unimplemented data.
- Write the GLAS_HDF data structure to the GLAS_HDF file
- Set the dimension scales
- Create HDF5 hard links from time dimension scales back to logical time group
- Close the rate group
- Write metadata
- Close metadata groups
- Close the GLAS_HDF file.
- Print status information

Figure 12-2 shows this sequence graphically.

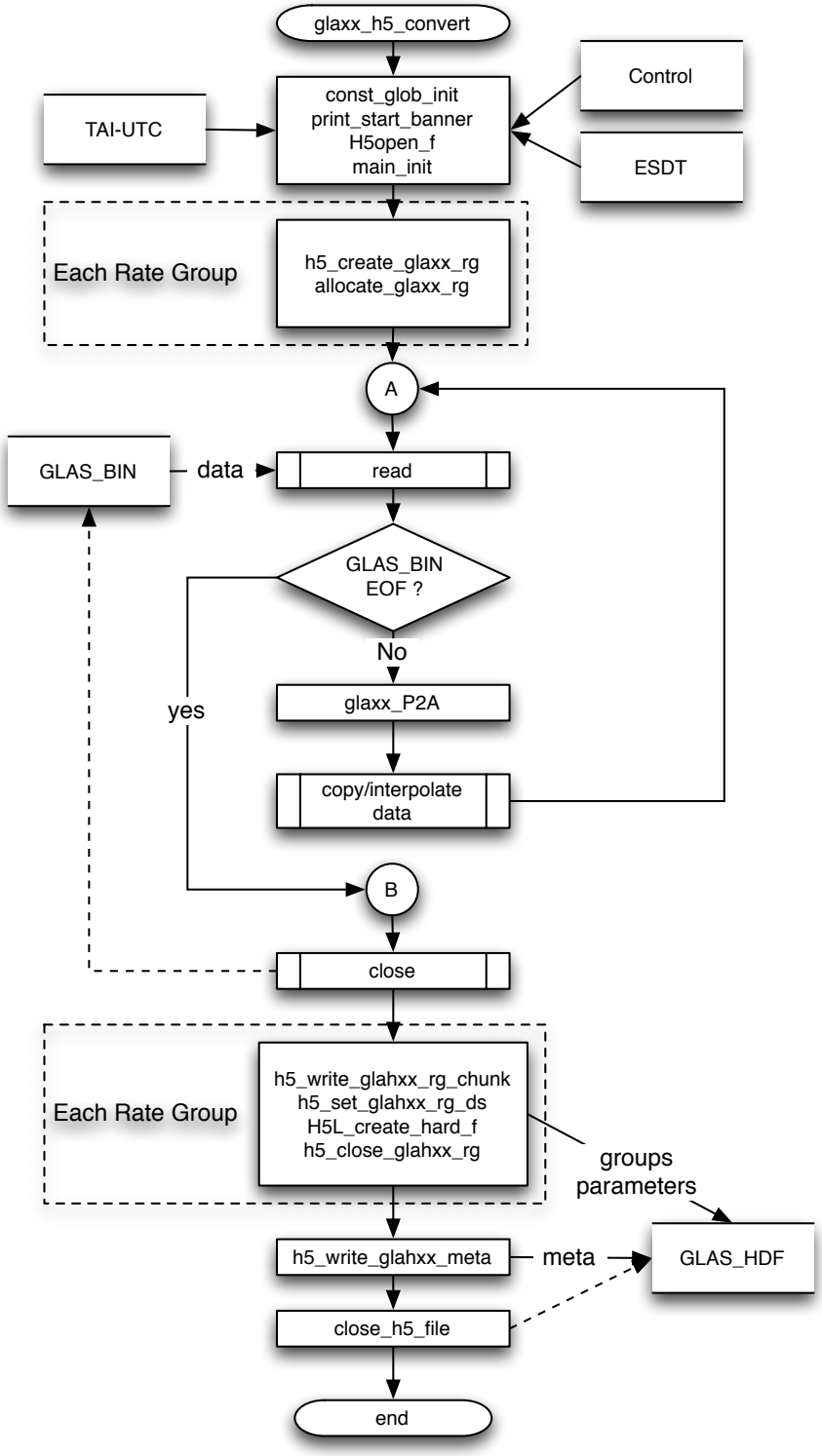

Figure 12-2 glahxx_h5_convert

12.3 Handling Multi-Rate Data

Most of the modifications a programmer must perform on the generated glahxx_h5_convert code involve copying multi-rate data from GLAS_BIN to GLAS_HDF data structures. GLAS_BIN "records" handle multi-rate data by using arrays within the record. GLAS_HDF "records" are temporally flat. For example, consider a theoretical array of measurements recorded at a 5 Hz rate. GLAS_BIN would store these as a 5-element array within a 1 second record. GLAS_HDF would store these as single elements within 5 Hz records. The difference in layouts is illustrated in Figure 12-3.

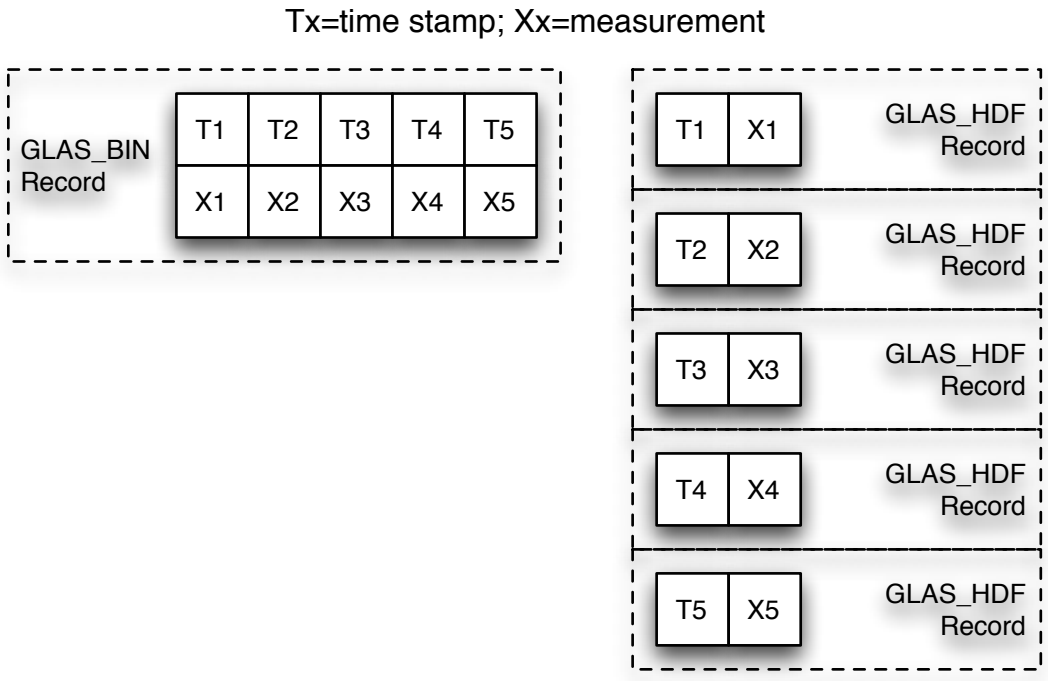

Figure 12-3 Data Storage Comparison

The transfer of multi-rate data is accomplished with a "flattening" technique. A simple loop is used to copy each GLAS_BIN element to the appropriate location in GLAS_HDF. In the infrequent cases where multi-rate data are required, math_lib supplies a couple of different mathematical routines to achieve the desired interpolation.

12.4 Product-Specific Model Deviations

Most of the product conversion PGEs directly follow the glahxx_h5_covert model. There are a couple of exceptions where deviations were necessary.

12.4.1 gla04_h5_convert

GLAH04 GLAS_HDF was defined as a single file. GLA04 GLAS_BIN files were implemented as multiple files. The gla04_h5_convert PGE was required to read multiple input GLAS_BIN files (one for each GLA04 subtype) and write a single GLAH04 file. This deviation was implemented in two major steps: 1) main_init was modified to accept multiple input files and 2) the data-copying portion of the gla04_h5_convert code was modified to copy data from the appropriate GLA04 data structure.

12.4.2 gla01_h5_convert

Waveforms are the most basic and useful measurement collected by the GLAS laser altimeter. Most of the primary science measurements are generated from waveforms. Waveform storage on the GLA01 GLA_BIN product was less than optimal, so additional requirements were leveled on GLAS_HDF to improve waveform accessibility.

In particular, the altimeter waveform does not fit well within the dimension scale concept. Sample location changes with surface and return signal condition. The waveform samples are a two-dimension array where samples are taken at intervals for 5 possible sample locations (ranges) within the waveform. The first dimension scale is time since the array is time-based. The second dimension scale corresponds to the sample locations within the waveform relative to the sample farthest from the spacecraft. However this changes based on compression type and a land/water mask, so the design uses an index to another array that is a dimension scale that matches the waveform dimension scale. Both the waveform samples and the sample locations are dimensioned with values 1 to 544. The waveform has the return signal level in volts for each sample 1 to 544 and an index to the sample locations first dimension (1 to 5). For a specific waveform the sample location (index,i) where i is the same dimension scale 1 to 544 as the waveform and has the distances in nanoseconds for each sample relative to the sample farthest from the spacecraft for that waveform.

13.0 GLAH_META COMPONENT

glah_meta is the PGE that extracts metadata from a GLAS_HDF file and writes it ECHO format to a detached metadata file (MET). Most of glah_meta is a reuse of the code that instantiated the GSAS glas_meta PGE. Since the metadata extraction process is generic, a single PGE is used for all of the GLAS_HDF filetypes. glah_meta is implemented as a main Fortran program (glah_meta) and an initialization module (main_init).

13.1 main_init

main_init provides initialization functions for glah_meta. It is virtually identical to the main_init routine described within glaxx_h5_convert and only two major differences are discussed here.

glah_meta must store QA information passed via control in each MET file. The QA information was originally generated by GSAS after a GLAS_BIN granule was produced, stored in a database and added to GLAS_BIN MET files by the GSAS glas_meta PGE. Since this information is not stored on the GLAS_BIN products, the same functionality is replicated by glah_meta. main_init parses the QA information from the glah_meta control files and returns it to glah_meta.

Likewise, NOSE information was provided within the GLAS_BIN MET files, but not within the GLAS_BIN files themselves. Ancillary PASSID information is required to generate NOSE data and this information is read from control and returned to glas_meta.

13.2 glah_meta

glah_meta is the Fortran program that implements the PGE. It is mostly a processing shell that calls library routines but does have logic to handle QA and NOSE information. Execution flows through glah_meta as follows:

- Initialize global constants
- Print status information
- Open the HDF5_LIBRARY
- Call main_init
- Parse information from the ESDT file.
- Read ancillary and grouped metadata from the GLAS_HDF file.
- Calculate the NOSE bin numbers.
- Read 1Hz time, lat, lon and attflg from the GLAS_HDF file.
- Calculate NOSE information.
- Write the MET file in ECHO format.
- Close the GLAS_HDF file.

- Print status information

This sequence is illustrated graphically in Figure 13-1.

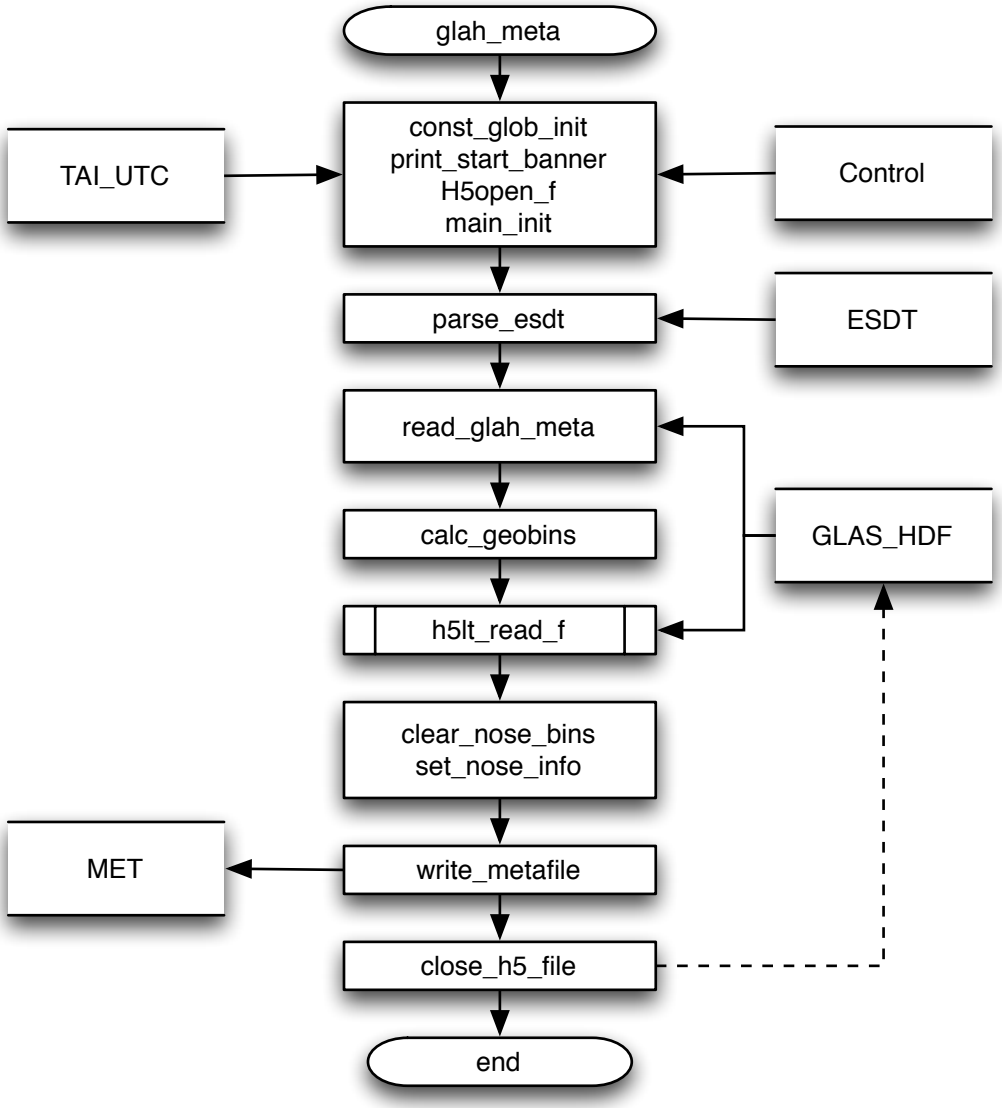

Figure 13-1 glah_meta

13.3 glah_meta Product Input

glah_meta requires 1Hz time, position, and flag data from the GLAS_HDF file to compute NOSE information. This is an excellent example where using the HDF5_LIBRARY H5LT interface is simpler and more efficient that using the glahxx_api interface. Using the glahxx_api would require reading all of the GLAS_HDF parameters when only four are actually necessary.

The following is a code fragment that shows how to use the H5LT interface to read a single element from a GLAS_HDF file:

```
call h5ltread_dataset_f(fs_in(1)%h5file_id, "/Data_1HZ/Time/d_UTCTime_1", &
      H5T_NATIVE_DOUBLE, UTCTime, dims, i_res)
```

14.0 GLAH_BRW COMPONENT

The glah_brw process is instantiated as a single IDL program, hdf2hdf5.pro. It copies browse images from a HDF4-format GLAS_BIN browse (BRW) file into a /BROWSE group on the corresponding GLAS_HDF product. The code is very generic and can easily be modified for other similar purposes. (It was actually submitted as example code to the HDFGroup after support couldn't provide an easy method to implement it.)

The design decision to implement this functionality in IDL was primarily driven by the fact that IDL supports both versions 4 and 5 of the HDF_LIBRARY. The other factor was that the GSAS BRW creation code was written in IDL and was available to re-use as part of hdf2hdf5.

A complication of the copy process is that the native image format changed between HDF4 and HDF5. The image format for HDF4 was an indexed color bitmap. The native image format for HDF5 is TrueColor. hdf2hdf5 implements this image conversion process fairly easily using IDL native functions.

The hdf2hdf5 processing sequence follows:

- Open input HDF4 BRW file.
- Open output GLAS_HDF5 (for modification, not rewrite).
- Create a /BROWSE group on the GLAS_HDF file.
- Get the number of images in the HDF4 BRW.
- For each image…
    - Read the image from the BRW file.
    - Convert the image to TrueColor.
    - Write the image as chunked data to the /BROWSE group on the GLAS_HDF file.
- Close the open files.

15.0 GLA_CODEGEN COMPONENT

gla_codegen is a utility PGE that creates glahxx_api code for a specific product type. The generated code is highly-specific to the MABEL/GLAS_HDF implementation of HDF5 products.

15.1 Developmental Considerations

glas_codegen was developed specifically for GLAS_HDF. The concept, however, was borrowed from a more simplistic shell script-based code-fragment generator used for MABEL. During MABEL product development, it was recognized that the routines required to instantiate different product types were nearly identical. The only major changes between two implementations were the names and attributes of the data parameters.

Another developmental consideration that made creation of gla_codegen attractive was that the GLAS product database contained the names, type, and dimensions of each parameter contained within a GLAS_BIN file. In addition, it contained a significant number of the values required to add CF attributes to the parameters. A code generator could leverage that content to make product API generation nearly automatic.

In the end, the sheer volume of code necessary to implement the product APIs and the availability of the GLAS Product Database made the gains of developing code-generation capability outweigh the cost.

15.2 Implementation

Much of the code-generation functionality is embedded in the h5_codegen module located within common_lib/hdf_lib. The h5_codegen module contains generic functions; gla_codegen provides GLAS_HDF-specific functions. The gla_codegen PGE writes a Fortran module for each rate group defined in the product specification.

15.2.1 h5_codegen

h5_codegen is essentially a collection of form-generator subroutines that write Fortran code based on data structures that describe the product format and content. The following code fragment illustrates the data type that contains grouping information.

```
type, public :: in_group_type
  character(len=MAXSTR) :: &
    gkey, &                  ! Group key
    glab, &                  ! Group label
    gcoords, &               ! Comma-separated list of coordinate variables
    ghertz, &                ! Data rate (hertz)
    gtimeparam               ! The time parameter
  character(len=MAXLINE) :: &
    gdesc                    ! Group description
  integer :: &
    rnum                     ! Index number of encompassing rate group
end type in_group_type
```

This group type is instantiated as two allocable arrays of grouping information forming linked lists. g_garr contains logical group information. g_rarr contains rate group information. Each logical group element contains a pointer to its encompassing rate group element.

The next code fragment illustrates the data type that contains parameter information.

```
type, public :: in_stru_type
  character(len=MAXSTR) :: &
    long_name, &              ! Short description/long name
    init_value, &             ! Initialization value
    a_name, &                 ! GSAS algorithm variable name
    cf_name, &                ! CF standard name
    a_dtype, &                ! GSAS algorithm data type
    a_h5type, &               ! HDF5 native data type
    a_h5outtype, &            ! HDF5 output data type
    att_sub, &                ! H5 attribute subroutine
    p_sub, &                  ! H5 read/write subroutine suffix
    a_dsize, &                ! GSAS algorithm data size
    slen, &                   ! GSAS algorithm character length
    a_scale, &                ! GSAS algorithm scale factor
    a_units, &                ! GSAS algorithm units
    a_min, &                  ! GSAS algorithm min
    a_max, &                  ! GSAS algorithm max
    a_flag_values, &          ! CF comma-separated flag values
    a_flag_meanings, &        ! CF space-delimited flag meanings
    a_invalid                 ! GSAS algorithm invalid value
  character(len=MAXLINE) :: &
    desc                      ! Description
  integer :: &
    gnum, &                   ! Index number of logical group
    rnum                      ! Index number of rate group
end type in_stru_type
```

The parameter type is instantiated as two allocable arrays that contain information for each parameter on the target product. One (g_full_var) contains all the parameters on the target product. The other (g_varr) contains only the parameters contained within the active group. Indexes are provided which link each parameter to a specific logical group and a specific rate group.

The final pieces of information needed by h5_codegen are the identifiers of the project for which the code is being generated and the file id of the data product.

Once each of these data structures have been filled with valid information, each of the h5_codegen subroutines can be called to write a specific piece of product API code. h5_codegen subroutines are listed in Table 15-1.

Table 15-1 h5_codegen Subroutines

Subroutine	Description
write_module_start	Writes the API module start and definitions
write_init	Writes the API data initialization subroutine
write_alloc	Writes the API allocation/deallocation subroutines

Subroutine	Description
write_open	Writes the API group open (for reading) subroutine
write_close	Writes the API group close subroutine
write_create	Writes the API group creation (for write) subroutine
write_read	Writes the API group chunked read subroutine
write_write	Writes the API group chunked write subroutine
write_print_attr	Writes the API parameter attributes print subroutine
write_print_head	Writes the API parameter header print subroutine
write_print_data	Writes the API parameter data print subroutine
write_data_dict	Writes the API data dictionary (HTML) subroutine
write_sync	Writes the API data synchronization subroutine.
write_setds	Writes the API dimension scale subroutine
write_end	Writes the end of API module

15.2.2 gla_codegen

gla_codegen is the GLAS_HDF program that fills the data structures required by h5_codegen and then calls the appropriate h5_codegen routines in sequence to write glahxx_api code. gla_codegen is implemented within a main program (gla_codegen) and a module (gla_codegen_mod).

gla_codegen_mod provides two major functions – parsing information from the product description to fill the h5_codegen data structures and creating the main programs for glaxx_h5_convert and glahxx_dd.

In a major reuse of existing code, the GLAS_HDF product descriptions are written within the context of a GLAS_HDF control file. The control file creation process is fully documented in Appendix C with the end result being the information that exists in the GLAS Data Dictionary transformed into the information necessary to fill the h5_codegen data structures. The resulting control file contains product-specific labeling information and then the product description. An example is shown below. (The product description lines are not shown here but detailed in Appendix C).

```
=gla_codegen
project=GLAS_HDF
product=GLAH15
gla_product=GLA15
=data-dict
…
```

gla_codegen_mod subroutines are listed in Table 15-2.

Table 15-2 gla_codegen_mod Subroutines

Subroutine	Description
parse_dtype	Parses the HDF5 datatype and datasize from input GSAS datatype.
read_prod_cntl	Reads and parses the gla_codegen control file.
write_main_init	Writes the glaxx_h5_convert main_init subroutine
write_main	Write the glaxx_h5_convert main program.
write_gen_dd	Writes the glahxx_dd main program.

gla_codegen calls routines from gla_codegen_mod and h5_codegen_mod to write the glaxx_api code. For safety, h5_codegen uniquely names all the output files by prefixing "autogen_" to the name. This is intended to prevent coders from accidently overwriting modified files.

The gla_codegen processing sequence follows:

- Read the product specification control file.
- Fill the h5_codegen data structures.
- Create, write and close the glahxx_dd.f90 file.
- Create, write and close the main_init_mod.f90 file.
- Create, write and close the glaxx_h5_convert.f90 file.
- For each rate_group (rg) in the product specification…
    - Create the glaxx_rg_mod.f90 file.
    - Create a subset of the parameters in the rate_group to fill g_varr
    - Call h5_codegen subroutines to write subroutines within glaxx_rg_mod.f90
    - Close the glaxx_rg_mod.f90 file.
- Deallocate data structures.

This is illustrated graphically in Figure 15-1.

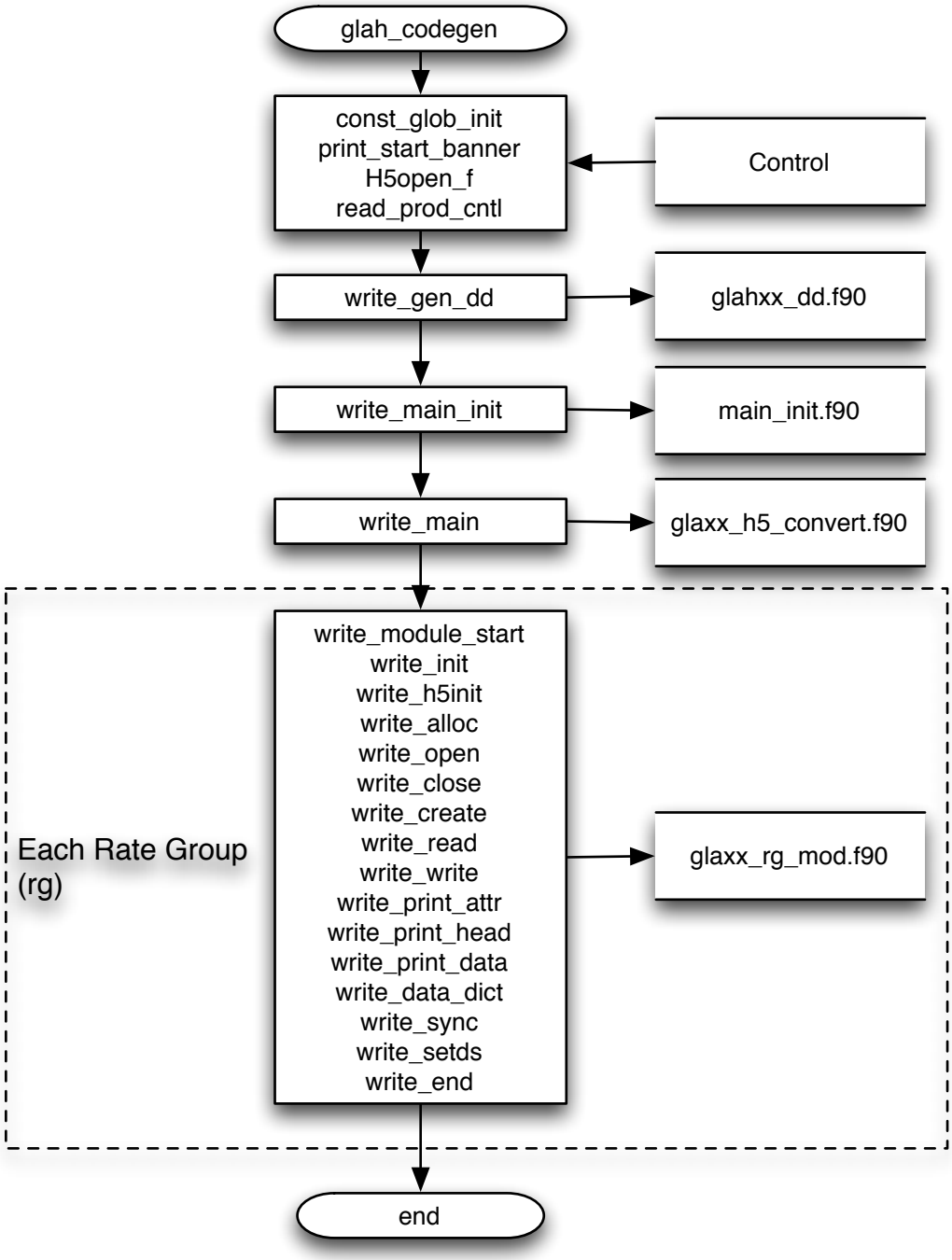

Figure 15-1 gla_codegen

.

APPENDIX A. REQUIREMENTS TRACE

REQUIREMENTS

The following table traces each GLAS_HDF requirement to sections within this document that describes an implementation that satisfies the requirement.

Identifier	Requirement	Trace
REQ_GLAS_HDF_001	The software shall transform GLAS integer-binary products into a standards compliant format.	
REQ_GLAS_HDF_001.1	The software shall use HDF5 as the standard data product file format (ESDS-RFC-007).	
REQ_GLAS_HDF_001.2	The software shall create the products with HDF5 CF-compliant parameter attributes to make the products self-documenting. This will allow data dictionaries to be created directly from the products themselves.	
REQ_GLAS_HDF_001.3	The software shall create the products with NetCDF-compliance in mind. This may allow the products to be used with NetCDF/HDF tools.	
REQ_GLAS_HDF_001.4	The software shall use compression where possible to decrease the size of the products.	14.0
REQ_GLAS_HDF_001.5	The software shall perform only transformation processes. No new science parameters shall be created.	12.2
REQ_GLAS_HDF_001.6	The software shall not put “spare” or un-implemented parameters on the products.	12.2
REQ_GLAS_HDF_002	The software shall make efforts to improve the usability of the products.	
REQ_GLAS_HDF_002.1	The software shall create products in such a manner that individual data values may be independently read.	
REQ_GLAS_HDF_002.2	The software shall logically group parameters, but at a level where desired data are not hidden.	
REQ_GLAS_HDF_002.3	The software shall transform the parameters from scaled-integer units into scientific units.	12.2
REQ_GLAS_HDF_002.4	The software shall provide a mechanism whereby each instance of a parameter can be associated with a time stamp and a	

Identifier	Requirement	Trace
	laser shot number.	
REQ_GLAS_HDF_002.5	The software shall store variable-rate waveforms in volts and provide relative sample times that will enable easy decompression of the waveforms.	
REQ_GLAS_HDF_002.6	The software shall incorporate multi-rate data within the same product.	12.3
REQ_GLAS_HDF_002.7	The software shall incorporate existing browse information into the products where available.	
REQ_GLAS_HDF_002.8	The software shall unpack bit flags where existing unpack routine already exist.	12.2
REQ_GLAS_HDF_003	The software shall incorporate metadata into the products.	
REQ_GLAS_HDF_003.1	The products will incorporate both human-readable and computer-parseable metadata.	
REQ_GLAS_HDF_003.2	The software shall support the same method of metadata exchange with NSIDC as the GLAS_BIN products. (External .MET files in ECHO format.)	
REQ_GLAS_HDF_003.3	The software will store lineage metadata on the products such that prior processing information is not lost.	
REQ_GLAS_HDF_003.4	The software shall support product-level digital object identifiers (DOIs) as defined by the ESDIS pilot DOI effort.	
REQ_GLAS_HDF_003.5	The software shall support UUIDs as granule-level unique identifiers to extend the ESDIS pilot DOI effort.	
REQ_GLAS_HDF_004	The software shall re-use existing software to the maximum extent possible.	
REQ_GLAS_HDF_004.1	The software shall re-use existing GLAS Science Algorithm Software (GSAS).	
REQ_GLAS_HDF_004.2	The software shall re-use existing MABEL Science Algorithm Software.	15.0
REQ_GLAS_HDF_004.3	The software shall be written to interface with re-used I-SIPS SDMS middleware for data management and job control.	

APPENDIX B. DIRECTORIES, MAKEFILES AND COMPILATION

DIRECTORIES

A directory tree of the GLAS_HDF source code and a description of each directory follows.

Directory	Description
.	Contains the master Makefile
./bin	Contains the common_libs and glas_hdf PGEs and utilities.
./idl	Contains generic IDL code.
./include	Contains include files for Makefiles.
./common_libs	Contains the common_libs Makefile
./common_libs/bin	Contains the common_libs PGEs and utilities.
./common_libs/doc	Contains common_libs documentation.
./common_libs/lib	Contains common_libs static libraries.
./common_libs/modules	Contains common_libs Fortran .mod files.
./common_libs/src	Contains common_libs/src Makefiles
./common_libs/src/anc_lib	Contains anc_lib Fortran code and Makefile.
./common_libs/src/cntl_lib	Contains cntl_lib Fortran code and Makefile.
./common_libs/src/const_lib	Contains const_lib Fortran code and Makefile.
./common_libs/src/err_lib	Contains err_lib Fortran code and Makefile.
./common_libs/src/hdf_lib	Contains hdf_lib Fortran code and Makefile.
./common_libs/src/math_lib	Contains math_lib Fortran code and Makefile.
./common_libs/src/mutil_lib	Contains mutil_lib Fortran code and Makefile.
./common_libs/src/time_lib	Contains time_lib Fortran code and Makefile.
./common_libs/src/util	Contains common_libs/util Makefile
./common_libs/src/util/minmax	Contains minmax utility Fortran code and Makefile.
./common_libs/src/util/timeconv	Contains timeconv utiltiy Fortran code and Makeilfe.
./glas_hdf	Contains glas_hdf Makefile.
./glas_hdf/bin	Contains glas_hdf PGEs and utilities.
./glas_hdf/data	Contains static ancillary files.
./glas_hdf/data/esdts	Contains ESDT files.
./glas_hdf/doc	Contains glas_hdf documentation.
./glas_hdf/idl	Contains glas_hdf IDL code.
./glas_hdf/lib	Contains glas_hdf static libraries.

Directory	Description
./glas_hdf/modules	Contains glas_hdf Fortran .mod files.
./glas_hdf/src	Contains glas_hdf/src Makefile.
./glas_hdf/src/gla01_hdf5	Contains gla01_hdf Fortran code and Makefile.
./glas_hdf/src/gla02_hdf5	Contains gla02_hdf Fortran code and Makefile.
./glas_hdf/src/gla03_hdf5	Contains gla03_hdf Fortran code and Makefile.
./glas_hdf/src/gla04_hdf5	Contains gla04_hdf Fortran code and Makefile.
./glas_hdf/src/gla05_hdf5	Contains gla05_hdf Fortran code and Makefile.
./glas_hdf/src/gla06_hdf5	Contains gla06_hdf Fortran code and Makefile.
./glas_hdf/src/gla07_hdf5	Contains gla07_hdf Fortran code and Makefile.
./glas_hdf/src/gla08_hdf5	Contains gla08_hdf Fortran code and Makefile.
./glas_hdf/src/gla09_hdf5	Contains gla09_hdf Fortran code and Makefile.
./glas_hdf/src/gla10_hdf5	Contains gla10_hdf Fortran code and Makefile.
./glas_hdf/src/gla11_hdf5	Contains gla11_hdf Fortran code and Makefile.
./glas_hdf/src/gla12_hdf5	Contains gla12_hdf Fortran code and Makefile.
./glas_hdf/src/gla13_hdf5	Contains gla13_hdf Fortran code and Makefile.
./glas_hdf/src/gla14_hdf5	Contains gla14_hdf Fortran code and Makefile.
./glas_hdf/src/gla15_hdf5	Contains gla15_hdf Fortran code and Makefile.
./glas_hdf/src/glah_meta	Contains glah_meta Fortran code and Makefile.
./glas_hdf/src/gsas_lib	Contains gsas_lib Fortran code and Makefile.
./glas_hdf/src/util	Contains utility Makefile.
./glas_hdf/src/util/glas_codegen	Contains glas_codegen Fortran code and Makefile.

MAKEFILES

GLAS_HDF code is built using cascading Makefiles. The Makefiles force the required order of compilation. However, since common_libs is compiled as a static library and each PGE is independent of the others, a developer can build the complete system at the top level and then use the local Makefile to compile only the code he is modifying.

GLAS_HDF Makefiles are configured to support the Linux/gfortran development environment. However, by modifying the ./glas_hdf/include/make_defaults.incl file, a developer can customize the compiler, link options, and compiler flags.

BUILDING

To compile common_libs and glas_hdf, start at the top level of the development tree and clean the development tree by typing ‘make clean’. Build the development tree by typing ‘make’.

APPENDIX C. GLAS_HDF PRODUCT DEVELOPMENT PROCEDURES

SPREADSHEET DEVELOPMENT

Submit a Jira item for product development.

Export the information from the GLAS Product Database into an Excel spreadsheet.

Remove the 'spare' fields.

Create the rate and logical groups. Be sure to add a control file keyword at the front and a description at the end. Use "/" to distinguish subgroups. Each rate group within the spreadsheet must contain the following information:

- Column A – Keyword and Group identifier (e.g.: RateGroup=d1)
- Column B – Group Label (e.g.: Data_1Hz)
- Column C – Coordinate parameter(s) (e.g.: DS_UTCTime_1)
- Column D –Time variable (e.g.: DS_UTCTime_1)
- Column E – Data Rate in Hz (e.g.: 1)
- Column P – Description

At least one Dimension Scale line follows each rate group. The line has the keyword "DimensionScale=" in column A. The rest of the row is formatted exactly like a parameter description row.

Each logical group within the spreadsheet must contain the following information:

- Column A – Keyword and Group identifier (e.g.: RateGroup=d1)
- Column B – Group Label (e.g.: Data_1Hz)
- Column P – Description

Split out the packed flags and give them meaningful names.

Add/Update the following columns in each parameter row. flag names

- flag_meanings
- flag_values
- cf standard names
- units compliant with UDUNITS
- descriptions that need wording changes.

Verify that each variable datatype is one of the following. Anything else will not work with the code generator and must be handled manually.

- i1b, i2b, i4b, r4b, r8b, char

Verify that each invalid value is one of the following and represents scientific units. Anything else will not work with the code generator and must be handled manually. There is no mechanism in HDF for associating a flag variable with a field. Use "not_set" for those that don't make sense.

- invalid_i1b, invalid_i2b, invalid_i4b, invalid_r4b, invalid_r8b, (blank) | not_set

Convert “min” and “max” in terms of scientific units. Put 'not_set' where min/max does not make sense.

Save the spreadsheet to the repository.

CODE DEVELOPMENT

Compile the glas_hdf code:

```
cd <workspace>
 make clean
 make
```

If nothing failed you should be left with (at least) the code generator binary in the workspace bin directory : <workspace>/bin/glah_codegen

cd to the directory associated with your target product. example: <workspace>/glas_hdf/src/gla05_hdf.

Step 2. Verify Spreadsheet

Verify that the Excel spreadsheet is correct. It is very important to do this at the beginning since errors in the spreadsheet will only show up after you have done a lot of work on step 3 getting to step 4. The fewer time you get kicked back to this point, the smoother things will be.

Step 3. Create Control File

To begin code development, download the Excel version of the spreadsheet from the repository. Open it in Excel and Save As tab-delimited text file. IF NECESSARY, convert the carriage returns (CR) to linefeeds (LF). Rename the spreadsheet to something like glah06_format.ctl (or glah13_format.ctl, etc.).

Edit the spreadsheet (now – control file) and insert the required control-file breaks and keyword/value fields at the top. The section breaks delineate the control files-style variables definitions from the product description. Required variable definitions are the "project" and "product". CAPITALIZATION counts. The project should be "GLAS_HDF" and the product should be "GLAHxx". Make sure any version information or column title lines are commented out.

See example below:

Control File-style sections

```
=gla_codegen
 project=GLAS_HDF
 product=GLAH06
 gla_product=GLA06
 =data-dict
```

```
 # Version       20120423-1      jelee1
 #Keyword        Group Label     CF Standard Name        Product Var Name (not
used)     Long Name       Name    Datatype        Algorithm Scale Units   Min
Max     Invalid Implementation Note     flag_values     flag_meanings
Description
```

Be aware that the code generator currently supports only rank=1 and rank=2 arrays (I don't really think this is an issue with GLAS data...but maybe atmosphere?). Also be aware that every multi-dimension parameter MUST has an associated dimension scale. For example, on GLAH07, i_g_mbscs is dimensioned (548). The 548 represents the number of profile bins from 40 to -1 km for 532. So you would need to create a new dimension scale at the root level called "DS_40_m1km" whose description would be "532 profile bins from 40 to -1 km." That would be an integer dimensioned(548) and have values from 1-548. Since the same 548 is used other places, you only need to create the instance once and then you can write custom code to assign the 548 to the correct parameters.

Step 4. Run gla_codegen

This assumes you are in the directory associated with your target product (step 1).

Run the code generator:

```
../../bin/gla_codegen <cntl_file>
```

If an error occurs, revert to Step2. Otherwise, if it worked, you will be left with 5 modules of generated code, for GLAH06, these modules were

autogen_GLA06_h5_convert.f90
autogen_GLAH06_d1_mod.f90
autogen_GLAH06_d40_mod.f90
autogen_GLAH06_dd.f90
autogen_main_init_mod.f90

Rename these modules by removing the "autogen_" part of the name.

Create a Makefile (best practice = copy/modify the one from the gla06_hdf subdirectory)

Try to compile by typing 'make'

Step 5. Fix legacy GSAS

Fix the legacy GSAS code so that it will compile. All the required legacy GSAS code should be copied as AccuRev versioned item into its respective target directory. You can use AccuRev to look at the changes to the GLA06 code to get an idea of what needs to be fixed. If this is taking longer than a half-hour to complete, something is wrong. The following is an example of change required for GLAH13:

++ GLA13_prod_mod

Replace the module definitions with

```
 use kinds_mod
 use const_gsas_mod
 use c_compare_mod
 use textutil_mod
```

```
use singleline_mod
```

++ GLA13_hdr_mod

Get rid of all the pre-processor directives (e.g.: #ifdef...)

Replace the module definitions with

```
use kinds_mod
use const_glob_mod
use const_gsas_mod
use common_hdr_mod
use prod_def_mod
use error_mod
use keyval_mod
use anc45_meta_mod
```

Search and replace "fstruct_sub_type" with "fstruct_type"

Change: call parse_keyval(header(i_start:i_end),keyval)

To: call parse_keyval(header(i_start:i_end),keyval, ErrorSeverity)

++ GLA13_alg_mod

Replace the module definitions with

```
use kinds_mod
use const_gsas_mod
use c_compare_mod
use textutil_mod
use singleline_mod
```

++ GLA13_scal_mod

replace: use const_glob_mod

with: use const_gsas_mod

add: use textutil_mod

Step 6. Update the autogen code

The next step is updating the autogen code. WITH ANY LUCK, you will only have to touch the main program (gla06_h5_convert) and the rate group routines (e.g.: glah06_d1_mod.f90 and glah06_d40_mod.f90).

Within the rate group routines, the majority of the changes involve support for additional DimensionScales. You can use AccuRev to compare versions of GLAH13, 14, or 15 to see the changes that were necessary (GLAH06 is a bad example in this case because it was the guinea pig product during development).

For the main program, this is where all the logic of converting the data from GLA to GLAH is done. Look at the example routines for GLA13, 14, or 15. Things you should look for include:

Creating additional loop(s) within the main processing loop to flatten the mult-rate data.

Setting all GLAH flags to their appropriate GLA equivalent.

Using the interpolation routines to interpolate any parameters that need interpolation.

Adding to code to set the shot counter (for all rate groups)

Uncommenting the example code to create a hard link from the dimension scale times back to the Time group. (for all rate groups)

Step 7. Compile

When you compile, you may need to refresh your workspace in order to see changes merged by other developers. To do this, in the AccuRev GUI, click the green lightning bolt in the upper left side of the toolbar.

Then, rebuild everything by doing this:

```
cd <workspace>
make clean
make
```

To compile your conversion program, copy a Makefile from an existing conversion program (i.e.: gla06_hdf5/Makefile). Change the references from GLA06/GLAH06 to the appropriate fileid. Add/delete any necessary rate group modules.

Then,

```
make clean
make
```

Step 8. Add Sources to the Depot

Use the AccuRev GUI to select each newly-created source file (This is most likely only the files you renamed from autogen*) and the Makefile. Select "Actions->Add to Depot". You now have saved, versioned copies of your source code in the AccuRev depot. Anytime you wish to keep a local version of the file you are working on, from the AccuRev GUI, select the file and then "Actions->Keep". You may have to set the Search filter (on the bottom left of the GUI) to display "External" files to initially see the files you are adding.

Step 9. Promote Sources to Your Working Stream

Files in an AccuRev workspace reside within local disk storage. To allow others to see your work, you need to promote the source code to your working stream. To promote a file to your working stream, select the file within the AccuRev GUI and select "Actions->Promote". You may need to alternately set the Search filter (on the bottom left of the GUI) to display "Pending" and "Modified" files to see what changes need to be promoted. There is currently a problem with AccuRev/Jira integration. When you promote, you will be asked to select a Jira item to associate the changes with. At this point, just select whatever item is closest to what you are working on. Hopefully this will be fixed soon.

Step 10. Merge to Integration Branch

You should only merge to the integration branch once approval has been given by the CCB. Merging is as simple as dragging the change packages (visible in the stream browser) from your working stream to the integration stream.

APPENDIX D. GLOSSARY AND ACRONYMS

A2P	Algorithm-to-Product Conversion
ALT	Altimeter or Altimetry, also designation for the EOS-Altimeter spacecraft series
ANCxx	GLAS Ancillary Data Files
API	Interface used by software components to communicate with each other.
APID	GLAS Level-0 Data file
ATBD	Algorithm Theoretical Basis Document
ATM	Atmosphere
CCB	Change Control Board
DAAC	Distributed Active Archive Center
EDOS	EOS Data and Operations System
ELEV	Elevation
EOS	NASA Earth Observing System Mission Program
EOSDIS	Earth Observing System Data and Information System
GB	Gigabyte
GLAS	Geoscience Laser Altimeter System instrument or investigation
GLAxx	GLAS Science Data Product Files
GLAHxx	GLAS Science Data Product Files reformatted into HDF5.
GPS	Global Positioning System
GSAS	GLAS Science Algorithm Software
GSFC	NASA Goddard Space Flight Center at Greenbelt, Maryland
GSFC/WFF	NASA Goddard Space Flight Center/Wallops Flight Facility at Wallops Island, Virginia
HDF4	Hierarchal Data Format Version 4
HDF5	Hierarchal Data Format Version 5
HDF-EOS	EOS-specific Hierarchical Data Format
I-SIPS	Icesat Science Investigator Led Processing System
I/O	Input/Output
ICESAT	Ice, Cloud and Land Elevation Satellite
ID	Identification
IEEE	Institute for Electronics and Electrical Engineering
KB	Kilobyte
L0	Level 0
L1A	Level-1A
L1B	Level-1 B

L2	Level-2
LASER	Light Amplification by Stimulated Emission of Radiation
LIDAR	Light Detection and Ranging
LPA	Laser Pointing Array
LRS	Laser Reference System
MB	Megabyte
MET	Detached metadata file
N/A or NA	Not (/) Applicable
NASA	National Aeronautics and Space Administration
NOAA	National Oceanic and Atmospheric Administration
NOSE	Nominal Orbital Spatial Extent
P2A	Product-to-Algorithm Conversion
PDF	Portable Document Format
PGE	Product Generation Executable
QA	Quality Assessment
SDMS	Scheduling and Data Management System
SDP	Standard Data Products
SSRF	Science Software Requirements Document
UTC	Universal Time Correlation
WF	Waveform